НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

УДК: 004.054

«До захисту допущено»
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“___” _____2019 р.

Дипломний проект
на здобуття ступеня бакалавра

з напряму підготовки *6.050101 «Комп’ютерні науки»*

на тему: « *Система автоматизації тестування фреймворків Web-порталу* »

Виконав: студентка *4* курсу, групи *ІС-з51*
Устенко Юлія Тарасівна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *старший викладач Новікова П.А.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Тєлишева Т.О.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент *доц. каф. ММСА ІПСА, к.т.н. Дідковська М.В.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студентка Устенко Ю.Т. _____
(підпис)

Київ – 2019 р.

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з шести розділів, містить 37 рисунків, 3 таблиці, 1 додатків.

У дипломному проекті реалізована тема: «Система автоматизації тестування фреймворків Web-порталу», метою якої є зменшення часу, необхідного на виконання та аналіз тестів тестувальником.

У розділі загальні положення були описані діяльності моделі з боку тестувальника та з боку користувача веб-порталу, варіанти використання, розглянуто та проаналізовано загальний підхід до тестування програмного забезпечення та автоматизації, та правильно поставлена задача із визначеними цілями та метою.

У розділі з інформаційного забезпечення були визначені вхідні та вихідні дані, була показана схема вхідних даних.

У розділі з математичного забезпечення було розраховано ефективність впровадження автоматизації у процес розробки веб-порталу.

У розділі з програмного забезпечення описані основні засоби розробки комплексу задач, висунуті вимоги до технічного забезпечення, обрано та обґрунтовано архітектуру програмного забезпечення.

У технологічному розділі описана інструкція користувача, проведені випробування та показані результати з висновками.

АВТОМАТИЗАЦІЯ, ФРЕЙМВОРК, ПАТЕРН, БЕЗПЕРЕРВНА ІНТЕГРАЦІЯ, SELENIUM.

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of six sections, containing 37 pictures, 3 tables, 1 application.

The diploma project implements the theme «Automated Testing Framework for Web Portal» whose purpose is to reduce time for executing and analyzing tests by QA specialist.

The General Terms section describes the model's activities from QA specialist side and web-portal user side, usage patterns, examines and analyzes the general approach to software testing and automation testing, and correctly addresses the objectives with the goals and objectives.

In the information support section, input and output data were defined, the input schema was shown.

In the section on mathematical support were calculated efficiency of automation solution implementation in web-portal development proccess.

The software section describes the main tools for developing a set of tasks, advanced technical support requirements, and chooses and builds the architecture of the software.

The technological section describes the user's manual of the test and shows the results with conclusions.

AUTOMATION, FRAMEWORK, PATTERN, CONTINUOUS INTEGRATION, SELENIUM.

ЗМІСТ

ПЕРЕЛІК УМНОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CI (Continuous Integration) – спосіб розробки програмного забезпечення, який полягає у виконанні частих автоматизованих складових проекту для якнайшвидшого виявлення та вирішення інтеграційних функціональних проблем.

ROI (Return On Investments) – фінансовий коефіцієнт, який ілюструє рівень прибутку або збитку бізнесу, враховуючи суму зроблених інвестицій.

XML (eXtensible Markup Language) – стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками.

SQL (Structured Query Language) – мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів.

Патерн – шаблон, спосіб вирішення задач проектування програмного забезпечення. Не є закінченим зразком, який можна безпосередньо транслювати в програмний код.

ВСТУП

Велика кількість програмного продукту випускається та використовується у будь яких сферах нашого життя: будь-то індустрія розваг, наукова сфера, бізнес, медицина, машинобудування чи навіть індустрія харчової промисловості – усюди використовується програмне забезпечення, яке, насамперед, поліпшує ефективність виробництва та зменшує людський вплив на процес виконання певної роботи.

З часом важче й важче стає слідування вимогам щодо функціональної складової систем, адже майже у всіх сферах життя людській роботі віддають перевагу роботизованим системам та системам, де вплив людини є мінімальним. Від роботи таких систем залежить працездатність та ефективність роботи і прибуток великих підприємств, а іноді й життя людей. Ці фактори висувають наперед важливі питання щодо відмово стійкості, надійності та якості програмних забезпечень. З плином часу сам процес розробки програм змінювався: з’явилось багато нових методологій та інструментів, які дозволяють керувати, структурувати та прискорювати процеси розробки, удосконалювати оцінювання часу та ресурсів, призначених для отримання результату. Деякі методології навіть дозволяють швидко підлаштовуватись під постійно мінливі вимоги замовника під час розробки, це дає змогу ефективно оперувати та гнучко змінювати стратегію процесу.

При розробці продукту необхідно, щоб після випуску на ринок, він працював коректно, без збоїв, а також відповідав вимогам замовника. Також важливо, аби у процесі розробки команда розробників мала частково функціонуючий продукт з надійно працюючим та відповідним до вимог функціоналом, який з часом розширюється та удосконалюється. Це дозволяє, насамперед, вже з ранніх етапів розробки випускати продукт на ринок, що має прибуткову перевагу у зрівнянні з тими системами, які є повноцінно

функціонуючими лише у кінці розробки. Це завдання допомагають вирішити такий процеси тестування.

Тестування програмного забезпечення – процес аналізу програмного продукту та відповідної документації з метою виявлення дефектів та помилок і підвищення якості продукту[1].

Ця процедура допомагає розробникам об'єктивно зрозуміти, в якому стані зараз знаходиться система, що розроблюється, і, як наслідок, більш точно та ефективно планувати свої подальші дії.

Для розробки програмного продукту витрачається велика кількість коштів, кожен замовник хоче отримати якісний результат та покриття певної кількісті функціональності, тому впровадження тестування як однієї с основних частин розробки програми є особливо важливою та існує саме для забезпечення якості програмного продукту та підтримку робочого стану системи. Завдання зниження вартості розроблюємої програмної системи є одним із найактуальніших в індустрії інформаційних технологій[2].

Автоматизоване тестування дозволяє значно знизити витрати компаній замовників, зекономити ресурси та час, які використовуються для тестування та підтримки високої якості продукту, а також знизити ризик випуску на ринок неякісного продукту, чи продукту, який не задовольняє потреб користувачів.

Саме тому технології автоматизації тестування є досить популярними у інформаційно-технічних компаніях, робота яких пов’язана з розробкою програмних продуктів.

Дипломний проект присвячений розробці комплексу задач з автоматизації тестування веб-ресурсу, виявлення помилок і дефектів у обраній системі та покращення якості продукту шляхом створення фреймворку, який задовольнить усі потреби для тестування за допомогою комплекту інструментів Selenium та засобами Nunit.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

1.1.1 Поняття автоматизованого тестування і його значення для розробки програмного забезпечення

Одним з найбільш ефективних засобів контролю якості програмних продуктів є тестування. Тестування являється вагомою частиною розробки програмного забезпечення. Якістю у сфері розробки програмного забезпечення це не тільки зручність користування програмою або її надійність. Якість програмного засобу – це сукупність його властивостей, які є сукупність його придатності та задовольняють усі потреби за його призначенням [3]. Отже, якісним є продукт, який відповідає чітким критеріям якості, що пред'явлені до нього зацікавленими особами, користувачами або замовниками даного продукту. Щоб програмний продукт вважався якісним він повинен відповідати певним очікуванням ii стандартам. За визначенням Гленфорд Майерс, тестування – це процес дослідження програм з метою знаходження в них помилок [4]. У цьому визначенні під помилками, або дефектами програмного забезпечення, розуміють недоліки в розробленого програмного продукту, наслідком яких є їх невідповідність очікуваним результатам виконання програмного продукту і фактично отриманим результатам [5]. Тестування здійснюється шляхом проведення певних дій у тестованому додатку, результати виконання цих дій у подальшому звіряють з еталонними даними.

Також необхідно вказати, що існує поняття «принципів тестування». Основних принципів тестування 7, вони представлені нижче. Принципи тестування були розроблялися останні 50 років і є загальним збірником правил для тестування у цілому.

1) Тестування виявляє наявність дефектів

Тестування може виявити наявність дефектів в програмі, але не може довести їх відсутність. Важливо складати тест-кейси, які будуть знаходити максимальну кількість дефектів. Отже, при правильному тестовому покритті, тестування допоможе знизити ймовірність не виявлення дефектів в програмному забезпеченні. Якщо ж дефекти не були виявлені під час тестування, не можна з упевненістю стверджувати, що їх немає.

2) Повне тестування неможливе

Неможливо провести повне тестування, яке б враховувало усі комбінації вводу призначеного для користувача та станів системи, за винятком дуже примітивних випадків. Замість спроб виявлення всіх недоліків слід провести аналіз ризиків та розстановити пріоритети, що допоможе якомога більш ефективно розподілити зусилля по забезпеченню якості програмного забезпечення.

3) Раннє тестування

Тестування варто проводити напочатку життєвого циклу розробки програмного забезпечення. Зусилля тестування необхідно концентрувати на певних цілях.

4) Накопичення дефектів

Різні елементи системи можуть містити в собі різну кількість дефектів - тобто, частота виявлення дефектів в різних модулях програми може різнитися. Зусилля по тестуванню необхідно розподіляти пропорційно щільності дефектів. Як правило, найбільша кількість критичних дефектів розподілена в обмеженій кількості елементів системи. Що є проявом принципу Парето: 80% дефектів містяться в 20% модулів.

5) Парадокс пестициду

Виконуючи одні і ті самі тести знову і знову, Ви виявите те, що вони знаходять все меншу кількість нових помилок. Через те, що система еволюціонує, велика кількість раніше виявлених дефектів виправляються та старі тест-кейси більше не працюють.

Щоб запобігти виникненню цього парадоксу, необхідно час від часу вносити поправки в набори тестів, що використовуються, рецензувати і виправляти їх з метою досягнення їх відповідності новому стану системи і допомагали знаходженню якомога більшої кількості дефектів.

6) Тестування залежить від контексту

Вибір способу, техніки і виду тестування напряму залежить від самого програмного забезпечення. Наприклад, програмне забезпечення для військових потреб потребує більш суворої та ретельної перевірки, аніж, скажімо, комп'ютерна гра. З тих же міркувань, відповідальний сайт вимагає серйозного тестування на продуктивність, щоб перевірити можливість його роботи при високому навантаженні.

7) Помилка відсутності помилок

Факт того, що тестування не виявило помилок, ще не означає, що програма готова до користування. Пошук і виправлення дефектів будуть не важливими, якщо виявиться, що система незручна у використанні, і не задовольняє потреб користувача[6].

Тестування може проводитися як вручну, спеціалістами з тестування, так і автоматично, з використанням спеціальних програмних засобів. Процес верифікації програмного продукту, в ході якого основні етапи та функції тесту, такі як запуск, ініціалізація, виконання, аналіз і видача результату, проводяться

автоматично з допомогою інструментів автоматизованого тестування, і називається автоматизованим тестуванням програмного забезпечення [7].

У автоматизованого тестування є як свої переваги, так і недоліки. До сильних сторін автоматизованого тестування відносять:

- висока швидкість виконання, набагато перевершує можливості людини;
- відсутність впливу «людського фактора» (неуважність, втома);
- можливість багаторазового виконання тестів і зниження витрат ресурсів через це;
- виконання тест-кейсів особливої складності, недоступних людині;
- здатність зберігати, аналізувати в зручній формі великі обсяги даних;
- здатність виконувати низько-рівневі дії з додатком, з операційною системою тощо.

До недоліків автоматизованого тестування відносяться:

- необхідність залучення висококваліфікованого персоналу для автоматизації замість можливості використовувати працю ручних тестувальників;
- витрати на засоби автоматизації, на розробку і підтримку тестів;
- фінансові витрати та ризики, пов'язані з наявністю великої кількості наявних інструментів автоматизації і складністю вибору;
- старіння тестів у випадках змін вимог, переробки інтерфейсів продуктів, що тестуються.

Автоматизація тестування не дозволяє зовсім відмовитися від використання ручного тестування при розробці програмного забезпечення, але дозволяє суттєво знизити його частку, допомагає уникнути рутинних дій у процесі тестування, зменшити вартість і суттєво покращити якість програмного

забезпечення, що розроблюється. Тестування є однією з найбільш трудомістких фаз розробки програмного забезпечення, що займає багато часу. При цьому вартість виправлення помилки, не тільки матеріальна, а й фізична, як правило, безпосередньо залежить від часу, який проходить з моменту її виникнення до моменту виявлення. Вважається, що ідея автоматизації процесу тестування дозволить вирішити питання з нестачею часу для здійснення якісного тестування, ставала все більш актуальною та важливою по мірі збільшення складності розроблюваних програмних продуктів і зростання вимог до їх якості.

Зараз складно уявити собі розробку масштабного програмного проекту без використання автоматизованого тестування в процесі розробки. Етап розвитку тестування характеризується широкою інтеграцією тестування з процесом розробки в цілому, а також використанням автоматизації, великим набором технологій і інструментальних засобів та бібліотек для автоматизації[8].

1.1.2 Рівні автоматизації. Місце тестування користувальницького інтерфейсу в загальному процесі автоматизованого тестування

На сьогоднішній день існують різні підходи до автоматизації тестування. Загальна стратегія тестування включає в себе 3 рівні автоматизації:

1) рівень модульного тестування (Unit Tests Layer) - компонентні тести, зазвичай пишуться розробниками, тестується певна функціональність на етапі розробки перед впровадженням до працюючого коду;
2) рівень функціонального тестування (Functional Test Layer, or Service / API Tests Layer) - тестування через доступ до функціонального прошарку, минаючи призначений для користувача інтерфейс;

3) рівень тестування користувальницького інтерфейсу, або через призначений для користувача інтерфейс (GUI Test Layer) - дає можливість тестувати не тільки інтерфейс користувача, а й реалізовувати функціональне тестування, тобто виконувати операції, що використовують бізнес логіку програми.

Для забезпечення найкращої якості додатку рекомендується автоматизувати всі три рівня. На рисунку 1.1 наведена класична піраміда тестування[9].

Рисунок 1.1 – Піраміда тестування

Дана піраміда показує сбалансоване співвідношення тестів різного рівня для середньостатистичного проекту. Тести верхнього рівня вважаються більш складними в розробці і потребують більших матеріальних витрат. Співвідношення може змінюватися в залежності від специфіки продукту, що тестується.

1.1.3 Існуючі підходи до автоматизації. Патерн Page Object

На поточний момент на автоматизацію тестування веб-інтерфейсу користувача існує 4 основні підходи:

1) Capture / Playback – запис і відтворення. Даний підхід передбачає використання утиліт та програмних продуктів запису і відтворення, записуючих певну послідовність дій і потім її відтворюють вже без участі людини. Такі тести створюються легко і швидко, але за будь-яких змін функціональності чи інтерфейсу вимагають повного перезапису;
2) Scripting – написання сценарію. Методологія полягає у написанні тестових сценаріїв на мовах програмування, розроблених спеціально для автоматизації тестування. Даний підхід вирішує проблеми з масштабуванням і підтримкою тестів, відкриває великі можливості. Але при цьому є більш дорогим, що вимагає більшої кількості ресурсів, оскільки розробкою займаються спеціалісти більш високого рівня;
3) Data-driven testing – управління даними тестування. Методологія створення скриптів і верифікації їх на основі даних, що зберігаються у будь-якому сховищі або базі даних. Використовується у випадках, якщо необхідно реалізовувати однотипні види перевірок для множинних варіантів вхідних даних;
4) Keyword-based – Тестування за ключовими словами. Тест являє собою не програмний код, а послідовність дій з їх параметрами, описану за допомогою ключових слів. Це дозволяє створювати тести людям, які не мають навичок програмування.

Для довгострокових, складних проектів найбільш ефективними є друга і третя техніка (третя – при необхідності тестування на великому обсязі даних), вони вимагають більше ресурсів для реалізації, але при цьому окупаються в майбутньому, дозволяючи багаторазово використовувати як написаний код, так і самі створені тести (для регресійного тестування, включати їх в процес

безперервної інтеграції (CI), використовувати для навантажувального тестування, тощо). При написанні тестових скриптів зазвичай застосовують технологію фреймворків.

Фреймворк – це організація проекту, що дозволяє спростити розробку, модифікацію і підтримку програмного коду. У разі організації проекту для автоматизації тестування веб-інтерфейсу доводиться зважати на те, що тести потрібно підтримувати, а можливо і істотно переписувати в разі змін в інтерфейсі тестованої програми. Основне завдання, яке вирішується через використання фреймворків - зниження кількості змін, які потрібно внести в тести при змінах в тестованому додатку.

На даний момент в області автоматизації тестування існує один найбільш відомий шаблон проектування – Page Object. Подібна організація проекту дозволяє значно спростити підтримку тестів і знизити дублювання коду. Основні ідеї цього шаблону проектування:

– чіткий розподіл між безпосередньо кодом тестів і кодом, що спеціалізуються на роботі з локаторами (шляхами до елементів інтерфейсу) та веб елементами;
– наявність єдиного сховища для всіх служб і операцій, представлених на сторінці, а не безлічі розкиданих по всьому тестовому набору [10].

Патерн Page Object був розроблений для тестування веб-додатків, але його ідеї використовуються сьогодні у всіх інших видах автоматизованого тестування, існують різні різновиди реалізації даного шаблону. Основною перевагою його використання є те, що при змінах в інтерфейсі досить модифікувати код тестів тільки в одному місці (у розташуванні локаторів).

1.1.4 Існуючі підходи до автоматизації. Патерн Page Factory

Створення тест-кейсів може призвести до непідтримуваного проекту. Однією з причин є те, що використовується дуже багато дубльовання коду. Дубльований код може бути викликаний дублюванням функціональності, і це призведе до дублювання використання локаторів. Недоліком дубльованого коду є те, що проект є менш ремонтопридатним та підтримуваним. Якщо якийсь локатор зміниться, потрібно пройти весь код, щоб налаштувати локатори, де це необхідно. Використовуючи патерн Page Object разом з Page Factory, ми можемо зробити тестовий код, що є гнучким, і зменшити або усунути дублікати тестового коду. Реалізація об'єктної моделі сторінки може бути досягнута шляхом розділення абстракції тестового об'єкта і тестових скриптів.

Клас Page Factory в Selenium - це розширення патерну дизайну Page Object Page. Він використовується для ініціалізації елементів об'єкта сторінки або для створення самого об'єкта сторінки. Анотації до елементів також можуть бути створені (і рекомендовані), оскільки описуючі властивості можуть не завжди бути достатньо змістовними, щоб викликати один об'єкт з іншого.

1.2 Опис процесу діяльності

Фреймворк для автоматизованого тестування розроблюється для тестування бізнес логіки додатку та виконання певних тестів спеціалістом. Діяльність тестувальника у межах фреймворку направлене на створення тестових наборів даних, які подаються до системи у вигляді параметрів, описаних у текстовому документі та розширення тестових наборів у тих випадках, коли необхідно перевірити нову функціональність або додати нову валідацію, якщо попередня логіка була змінена. Класично тестувальнику необхідно виконати також ряд дій для виконання тестів: локально запустити тести та витрачати певний час на виконання нібито автоматизованого процесу і вже після цього при отриманні результатів або йти далі до наступних тестів, або

лагодити тести, які не пройшли та аналізувати причину. Схематично цей процес можна відобразити на діаграмі послідовності, який зображено на рисунку 1.2. Звісно, такий процес не дуже спрощує сам процес тестування, а спеціаліст витрачає майже стільки ж часу, скільки б було витрачено на ручне тестування. Даний проект дозволяє вирішити цю проблему за допомогою введення у процес тестування безперервної інтеграції та паралелізації виконання тестів. Це спрощує саме тестування та зменшує кількість витраченого часу, а також вартість виконання роботи. Користувач такої системи буде аналізувати пройдені тести за результатами, які подаються як вихідні дані процесу у вигляді звіту з параметрами пройдених тестів та тестів, які не пройшли.

Загалом, весь процес діяльності можна відобразити на діаграмі послідовності, що зображено на рисунку 1.3.

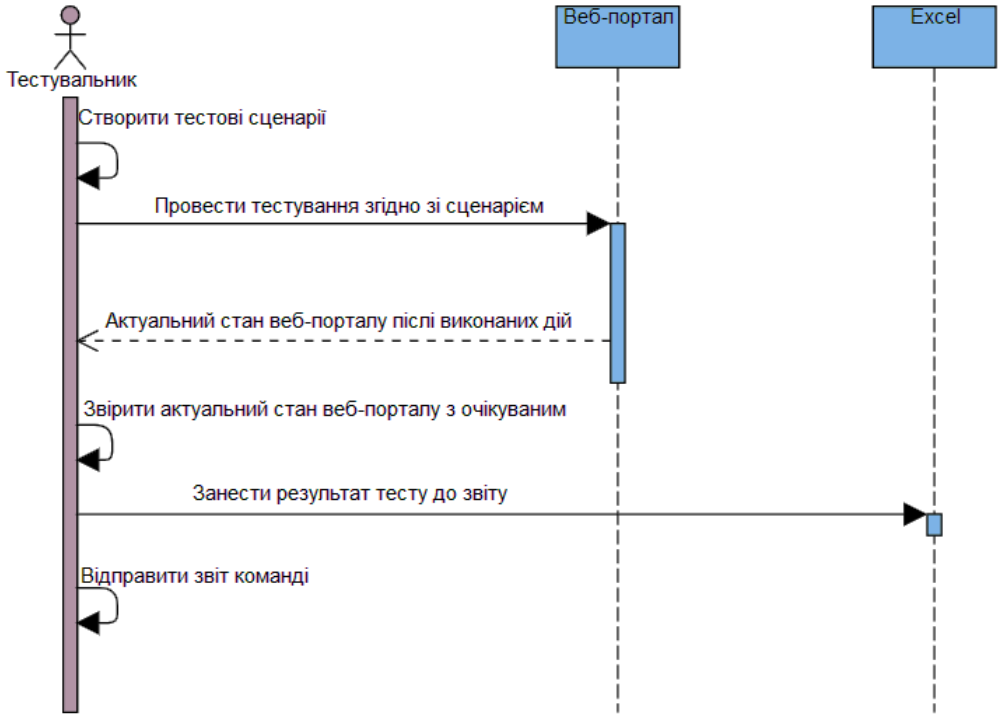


Рисунок 1.2 – Класична діаграма послідовності тестувальника

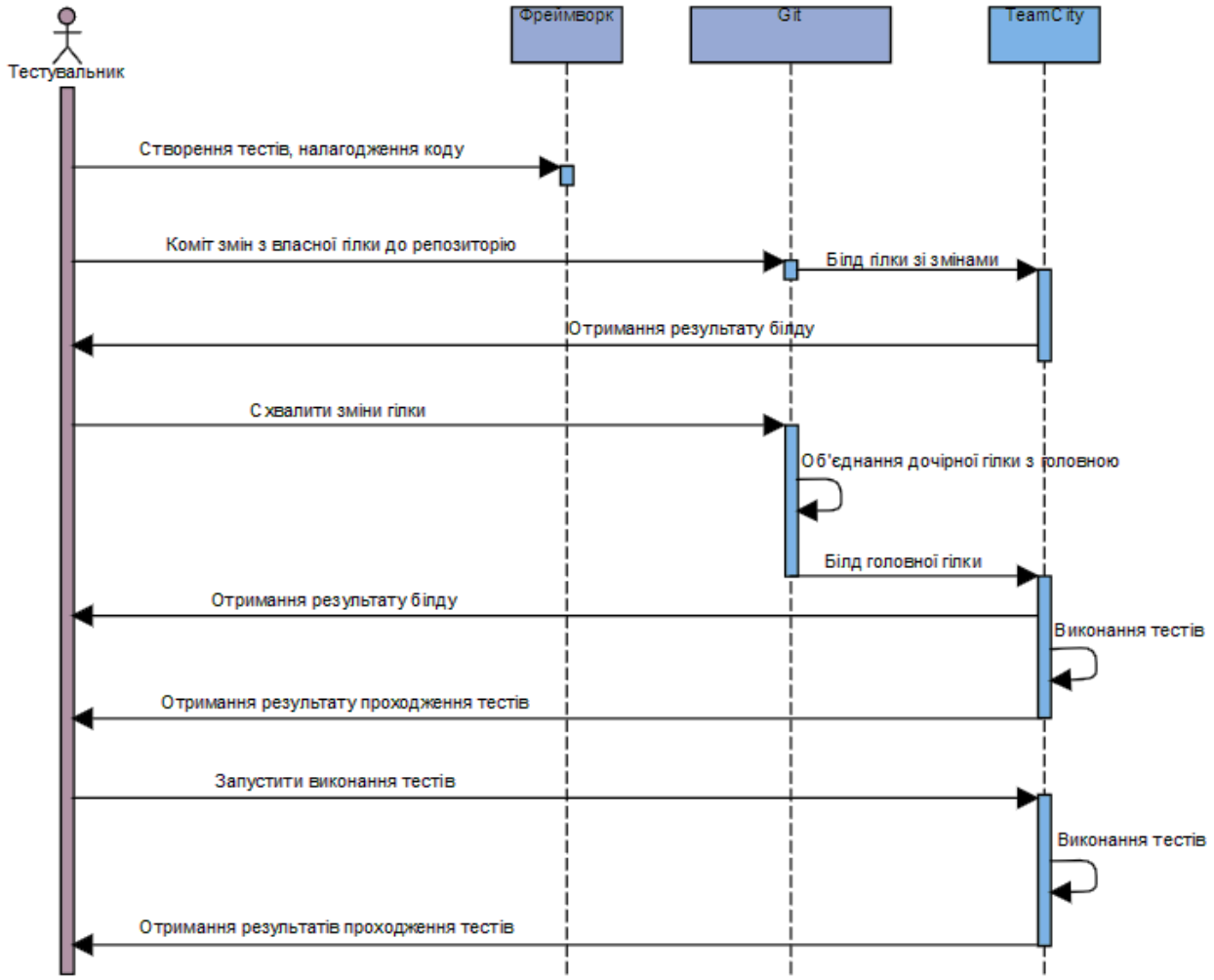


Рисунок 1.3 – Проектна діаграма послідовності

Розглянемо приклад діяльності веб-порталу «IT knowledge. Освітній IT кластер», який знаходиться на стадії розробки.

Перш за все, веб-ресурс призначений для об’єднання студентів, представників компаній та викладачів, задіяних в IT сфері для спілкування, навчання, надання та пошуку вакансій у різних технічних компаніях. Загально, кожен користувач може вести власний профайл, приклад використання зображено на рисунку 1.4, та створювати і переглядати пости, як зображено на рисунку 1.5. Студенти мають можливість спілкуватись з іншими користувачами, проходити кваліфікаційні тести, отримувати нові знання не

лише про технології, а й про різні ведучі компанії та відгуки щодо вищих навчальних закладів. Важливою функціональністю веб-порталу є створення резюме студентами для проходження практики та пошуку роботи у компаніях, які також зацікавлені у пошуку нових кадрів. Для представників компаній реалізована можливість створювати списки спеціальностей та відкритих вакансій, вони можуть відповідати на запити на підготовчу практику або на роботу, а також переглядати відкриті резюме студентів та запрошувати їх до компанії. У викладача університету є також можливість проходити практику та підвищувати кваліфікацію у різних IT компаніях, також створювати освітні тести по спеціальностям та створювати переліки тем та допоміжних матеріалів для курсових та дипломних проектів. Звісно, кожен з користувачів може, перш за все, переглядати різні матеріали на різні теми та створювати власні статті на наукову тему.

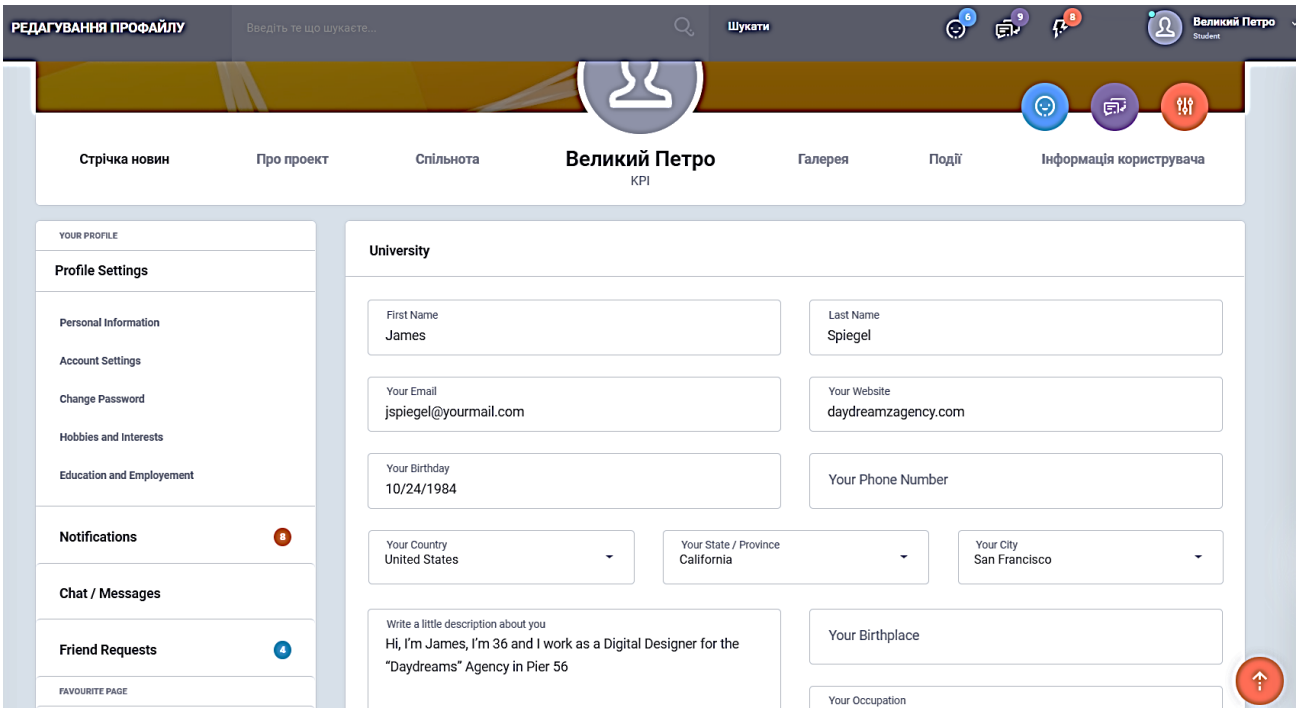


Рисунок 1.4 – Форма для ведення профайлу

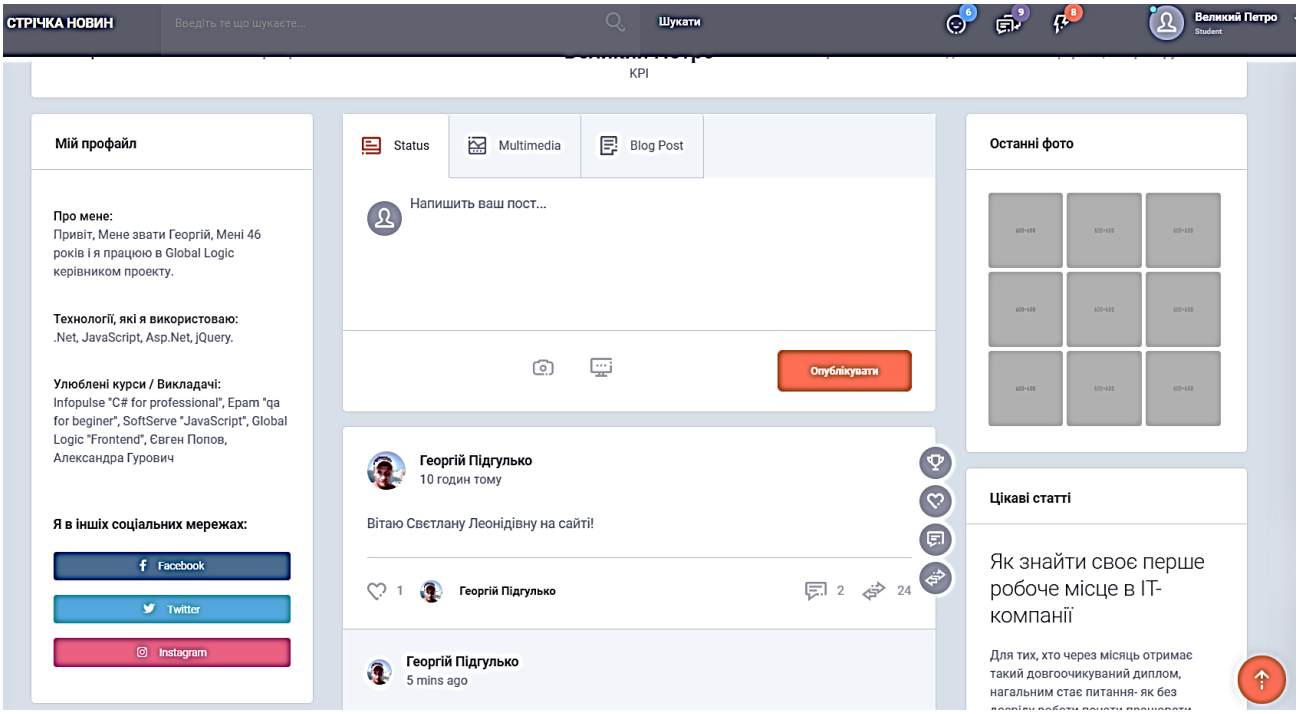


Рисунок 1.5 – Форма перегляду та створення постів

Як підсумок, веб-додаток, який піддається тестуванню, має такий функціональні пункти:

– реєстрація та вхід до порталу;
– ведення власного профайлу;
– створення та перегляд постів та коментарів;
– створення та перегляд наукових статей та інших матеріалів;
– проходження освітніх курсів;
– створення резюме, вакансій;
– ведення переліку тем для курсових та дипломних проектів.

Це графічно описується згідно з діаграмою прецедентів, зображеної на рисунку 1.6.

Рисунок 1.6 – Діаграма прецедентів

Даний перелік основних функцій є важливим для створення бази для функціонального тестування, який буде реалізовано у проектній роботі.

1.3 Огляд наявних аналогів

Найбільш відомими комерційними продуктами для автоматизації тестування користувальницького інтерфейсу веб- додатків є на сьогодні такі інструменти, як Serenity, Selenide, Cypress, Robot Framework, RedwoodHQ, Sahi, Gauge, Galen Framework, Citrus Framework.

В якості оціночних критеріїв при виборі інструментів рекомендується використовувати наступні[9]:

- підтримують кросплатформність (розпізнавання необхідних елементів інтерфейсу, технології взаємодії з ними, як реалізований процес взаємодії – задані параметри необхідні бути виконані на різних платформах, які є вимогою від замовника);
- підтримують типи додатків (нативні, веб, гібридні);
- підтримка запуску тестів на емуляторах, реальних пристроях;
- IDE (підтримують мови, зручність користування, інструменти налагодження);
- менеджери тестів (можливості по конфігурації тестових наборів, можливість паралельного запуску тестів, настройка параметрів запуску);
- логування, створення звітів (можливість надання звітів, налаштувань, інформативність).

При виборі інструмента фахівці радять брати до уваги специфіку додатки, поставлені завдання, кваліфікацію розробників та інженерів з тестування, технічні та фінансові ресурси, плани на майбутнє, ризики[11].

1.4 Постановка задачі

1.4.1 Призначення розробки

Метою даної роботи є зменшення часу, необхідного на виконання та аналіз тестів спеціалістом за рахунок впровадження автоматичного фреймворку для виконання тих самих дій.

1.4.2 Цілі та задачі розробки

Для досігнення даної мети повинні бути виконані такі задачі:

- розробка шаблонів автоматичних тестів;
- створення фремворку та інтеграція з системою управління версій Git;
- впровадження у систему безперервної інтеграції TeamCity процесу тестування та нотифікації.

Також варто зазначити, що досягти ідеально якісного продукту неможливо, згідно з принципами тестування. Тому дана робота призначена для досягнення 80% якості продукту, у відповідність до тестів.

Висновок до розділу

У даному розділі було розглянуто поняття тестування та детальніше автоматизоване тестування, недоліки та переваги використання автоматизації у проектах, та такі підходи для створення гнучких структур, як Page Object та Page Factory. Було визначено функціональну складову процесу діяльності, та створено основні пункти, за якими буде створено тестовий обсяг роботи. Також було проаналізовано ринок існуючих програмних аналогів та зазначені особливості, за якими краще обирати інструменти. Були визначені цілі та призначення дипломної роботи.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

До системи подається документ у вигляді XML файлу з параметрами, за якими буде виконуватись перевірка. Шаблони тестів отримують перелік параметрів та здійснюють перевірку саме того, що було описано у файлі. Такий підхід дозволяє абстрагувати тестові методи та виконувати багато перевірок, використовуючи лише параметри у файлі. Програмно це здійснюється за рахунок введення до ініціалізації веб-елементів атрибутів та класів та створення методів для обробки, які отримують елемент за його атрибутом та використовують для подальших перевірок.

Для створення тестових наборів даних необхідно враховувати, що перевірка усіх вхідних параметрів з усіма можливими вихідними даними у більшості випадків неможлива, так як їх кількість може бути надзвичайно великою. Як приклад, можна привести варіант перевірки поля для введення поштового адресу користувача. Як позитивний сценарій, можна ввести валідний адрес, як негативний – необхідно перевірити як система поведе себе у випадках, коли вводиться:

- неіснуюча адреса;
- адреса без доменого ім’я;
- SQL-ін’єкції;
- спеціальні символи;
- порожнє значення;
- значення, яке містить велику кількість символів, яка недопустима для поля;
- різні варіації із перерахованого вище.

А якщо ж один тест має перевірити ще й додатково поле з паролем, то кількість перевірок зростає удвічі. Для таких випадків передбачено

використання технік тест дизайну, які розповсюджено використовуються для створення тестових параметрів.

– еквівалентний поділ – наприклад, для поля введення паролю є умова, що кількість символів має бути від 6 до 16. При еквівалентному поділі необхідно вибрати одне вірне значення всередині інтервалу, скажімо, 10, і одне невірне значення поза інтервалу – 0;
– аналіз граничних значень – беручи приклад вище, в якості значень для позитивного тестування обирається мінімальна та максимальна межі (6 і 16), а також значення більше і менше кордонів (5 і 17) для негативного тестування. Ця техніка може бути застосована до полів, записів або до будь-якого роду сутностей, що мають обмеження;
– причина / слідство – це, як правило, введення комбінацій умов (причин) для отримання результату від системи (слідство). Наприклад, необхідно перевірити можливість входу до веб-ресурсу, використовуючи певну форму. Для цього треба заповнити поля «Поштова адреса» та «Пароль» і натиснути кнопку «Вхід» - це є причина. Після натиснення кнопки «Вхід» користувач бачить головне вікно сайту (будь-то стрічка новин, сторінка профайлу тощо) – це «слідство»;
– передбачення помилки – тестовий аналітик використовує свої знання системи і досвід до інтерпретації специфікації на предмет того, щоб визначити (або вгадати), за яких вхідних умовах система може видати помилку. Наприклад, специфікація каже: "користувач повинен ввести пароль для входу до системи". Спеціаліст буде

аналізувати так: "Що, якщо я не введу пароль?", "Що, якщо я введу невірний пароль?", тощо;

– повне тестування – це крайній випадок. В межах цієї техніки необхідно перевірити всі можливі комбінації вхідних значень, і вважається, що це повинно знайти всі дефекти. На практиці застосування цього методу не представляється можливим, через надзвичайно велику кількість вхідних значень.

Застосовуючи подані вище техніки, можна організувати тестові дані таким чином, щоб кількість перевірок була достатньо оптимальною, при цьому кожен важливий сценарій був покритий автоматизацією.

Для того, щоб користувач фреймворку мав можливість задавати різні параметри для проходження тестів, йому необхідно заповнити (або, якщо файл з даними вже існує, розширити чи оновити його) документ з XML розміткою, дотримуючись певних правил. Перш за все, дані повинні бути поділені між собою у блоках, а певні дані міститись у тегах. Також треба вірно вказувати назву тегу до параметру, який необхідно передати , інакше можна отримати невірну логіку виконання тестів або взагалі помилку компіляції.

До вхідного файлу необхідно додавати поля, перераховані у таблиці 2.1:

Таблиця 2.1 – Поля вхідного файлу

Блок	Назва параметру	Опис
userLogin		
	Email	Поштова адреса користувача
	Password	Пароль до акаунту
userRegistration		
	Role	Роль користувача
	Name	Ім’я користувача
	LastName	Прізвище користувача
	Institute	Інститут
	Position	Посада
additional	element	Елемент, до якого необхідна перевірка
	message	Повідомлення, яке необхідно отримати для перевірки

Використання тестових шаблонів та тестових наборів даних у файлах при створенні тестів дозволяє тестувальнику відмовитись від написання сценаріїв і коду, а використовувати певний шаблон для створення декількох перевірок одразу, модифікуючи тестові дані у вхідному файлі.

2.2 Вихідні дані

Вихідними даними у даному проекті служить звіт про виконання тестових сценаріїв. Для того, щоб зробити звіт про виконання навіть автоматизованих тестів, спеціалісту необхідно витратити певний час: створити файл (наприклад Excel), наповнити файл даними про тести (їх назву, статус, загальну кількість), та відправити кожному тестувальнику результати. Такий підхід є

ресурсномістким, так як, хоч тестувальнику не потрібно витрачати час на проходження тестів вручну, йому необхідно витрачати час на написання звіту, а значить, витрачаються й кошти замовника. Мій проект дозволяє вирішити цю проблему шляхом зв'язку фреймворку з сервісним інструментом TeamCity – сервісне програмне забезпечення для безперервної інтеграції та паралелізації виконання тестів; а на виході отримувати готовий звіт з усіма необхідними даними.

У тих випадках, коли тест не пройшов, користувач отримує дані щодо виконання у вигляді логованих даних. Це дозволяє більш точно знайти місце, де тест впав та швидше проаналізувати причину падіння та знайти дефект або у веб-додатку, або у фреймворку.

У звіті з результатами логовані дані зображено на рисунку 2.1:

```
Test.dll: Test.LoginPageFunctionality.LoginPage_CollapseMenu
  Login Menu is not collapsed!
  Expected: True
  But was:  False

   at Test.LoginPageFunctionality.LoginPage_CollapseMenu() in
C:\BuildAgent\work\feee03a2216e26a5\Test\LoginPageFunctionality.cs:line 52

------- Stdout: -------
[Info: 02:35:47 02.06.19] Click on Element for Button 'Expand Collapse Button'
(LoginHeaderSection.ExpandCollapseButton; JDI_Web.Selenium.Elements.APIInteract.GetElementModule)
Click on Element for Button 'Expand Collapse Button' (LoginHeaderSection.ExpandCollapseButton;
JDI_Web.Selenium.Elements.APIInteract.GetElementModule)
[Info: 02:35:48 02.06.19] Click on Element done
Click on Element done
[Debug: 02:35:48 02.06.19] Get Web Element: Context.Sections.LoginHeaderSection
[Debug: 02:35:48 02.06.19] OneElement found
[Info: 02:35:48 02.06.19] Is element displayed for Button '' (.;
JDI_Web.Selenium.Elements.APIInteract.GetElementModule)
Is element displayed for Button '' (.; JDI_Web.Selenium.Elements.APIInteract.GetElementModule)
[Debug: 02:35:48 02.06.19] Set wait timeout to 0
[Debug: 02:35:48 02.06.19] Set wait timeout to 20
[Info: 02:35:48 02.06.19] Get result 'True' in 0.03 seconds
Get result 'True' in 0.03 seconds
```

Рисунок 2.1 – Логовані дані тесту

На TeamCity результати виконання розташовані у вкладці Build Log, розташування зображено на рисунку 2.2, та містять усю інформацію, приклад зображено на рисунку 2.3:

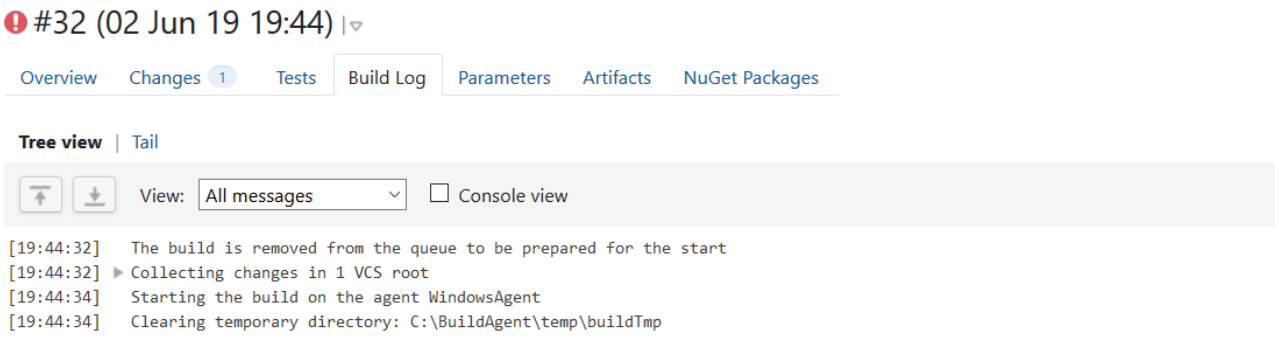


Рисунок 2.2 – Розташування вкладки Build Log

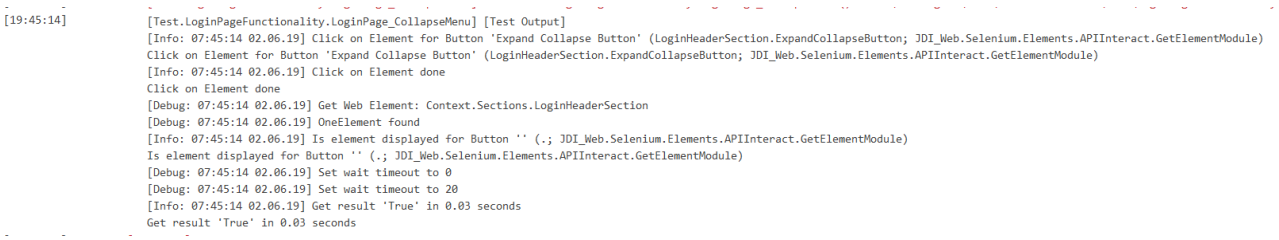


Рисунок 2.3 – Інформація про проходження тесту у TeamCity

2.3 Структура масивів інформації

Структура вхідних даних для тестів оформлюється у вигляді xml-розмітки, приклад зображено на рисунку 2.4:

```xml
<?xml version="1.0" encoding="utf-8"?>
<ListExtendedUsersWithMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <User>
    <Role>Student</Role>
    <FirstName>First</FirstName>
    <LastName>Last Name</LastName>
    <Password>password1</Password>
    <Email />
    <Institution>institu</Institution>
    <Object>
      <Item1>Email</Item1>
      <Item2>Це поле є обов'язкове.</Item2>
    </Object>
  </User>
  <User>
    <Role>Student</Role>
    <FirstName />
    <LastName>Last Name</LastName>
    <Password>password1</Password>
    <Email>1test@test.com</Email>
    <Institution>institu</Institution>
    <Object>
      <Item1>FirstName</Item1>
      <Item2>Це поле є обов'язкове.</Item2>
    </Object>
  </User>
  <User>
    <Role>Student</Role>
    <FirstName>Third</FirstName>
    <LastName>Last Name</LastName>
    <Password>password2</Password>
    <Email>2test@test.com</Email>
    <Institution />
    <Object>
      <Item1>Institution</Item1>
      <Item2>Це поле є обов'язкове.</Item2>
    </Object>
  </User>
  <User>...</User>
  <User>...</User>
</ListExtendedUsersWithMessage>
```

Рисунок 2.4 – Структура вхідних даних

Фреймворк отримує поданий файл з даними та модифікує його у колекцію з моделлю, з яким у майбутньому буде проходити тест. Модель класу за даними зображено на рисунку 2.5:

```
public class ExtendedUser
{
    [Name("User role")]
    0 references | 0 changes | 0 authors, 0 changes
    public UserRole? Role { get; set; }

    [Name("First name")]
    1 reference | 0 changes | 0 authors, 0 changes
    public string FirstName { get; set; }

    [Name("Last name")]
    0 references | 0 changes | 0 authors, 0 changes
    public string LastName { get; set; }

    [Name("Password")]
    2 references | 0 changes | 0 authors, 0 changes
    public string Password { get; set; }

    [Name("Email")]
    1 reference | 0 changes | 0 authors, 0 changes
    public string Email { get; set; }

    [Name("Position")]
    0 references | 0 changes | 0 authors, 0 changes
    public string Position { get; set; }

    [Name("Institution")]
    0 references | 0 changes | 0 authors, 0 changes
    public string Institution { get; set; }

    4 references | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public (string,string) Object { get; set; }
}

2 references | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
public class ListExtendedUsersWithMessage : ITestDataSource
{
    [XmlElement(ElementName = "User")]
    2 references | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public ExtendedUser[] Users { get; set; }
}
```

Рисунок 2.5 – Клас моделі вхідних даних

Висновок до розділу

В даному розділі було розглянуто схеми вхідних та вихідних даних, їх цінність для процесу тестування та можливість використання спеціалістом. Таким чином, відправляючи тестові набори даних до тестових шаблонів у вигляді файлу з XML-розміткою та отримуючи готовий звіт з усіма необхідними даними щодо виконання тестів, ми отримуємо автоматизований процес реалізації автоматизації тестування та спрощення класичних дій за допомогою інтеграції із програмними забезпеченнями, які надають необхідну функціональну складову для виконання тестів паралельно та звітування про виконання.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

Ціллю розробки є введення автоматизації процесу тестування веб-порталу на основі створення відповідного фреймворку та спрощення процесу отримання вихідних даних у вигляді виконаного звіту, обґрунтування доцільності використання для автоматизації тестування комерційних засобів розробки, виявлення сильних і слабких сторін такого рішення.

Для виконання поданих вимог необхідно вирішити такі задачі:

– проаналізувати області автоматизації тестування програмного забезпечення;
– вибрати інструмент для автоматизації тестування;
– створити гнучкий фреймворк для автоматизації тестування із подальшою можливістю корегування, розширення та удосконалення проекту;
– створити набір тестових шаблонів та розробити параметричні вхідні дані для тестів;
– реалізувати автоматичний запуск тестів на емуляторах і реальних пристроях;
– налаготити звітування спеціаліста по кожному з проходжень тестів.

3.2 Математична постановка задачі

Подані у попередньому підрозділі задачі є основною для мого проекту, але, дивлячись з боку реалізації та виконання цих цілей, постановкою задачі є такі пункти:

– розробити фреймворк тестування на основі шаблону JDI Framework, та удосконалити його для вирішення поставлених задач;

– використання інструментів та бібліотек Selenium та NUnit для можливості взаємодії з веб-драйвером браузеру та методик перевірки вхідних та вихідних даних;
– налагодження інтеграції с системою контроля версій Git для можливості контролю змін та керування доступом кожного із спеціалістів;
– інтеграція із системою безперервного виконання тестів TeamCity та налагодження тригерів для постійного їх виконання;

3.3 Обґрунтування методу розв'язання

Найбільш поширений спосіб оцінки ефективності (перш за все економічної) є розрахунок повернення інвестицій (ROI). Обчислюється він досить просто, відношенням прибутку до витрат.

У разі автоматизації під прибутком розуміється економія на ручному тестуванні.

Однак не завжди економія є метою впровадження автоматизації. Один із прикладів – швидкість виконання тестування (як по швидкості виконання одного тесту, так і по частоті проведення тестування). По ряду причин швидкість тестування може бути критичною для бізнесу - якщо вкладення в автоматизацію окупаються отриманим прибутком.

Інший приклад – виняток «людського фактора» з процесу тестування систем. Це важливо, коли точність і коректність виконання операцій є критичною для бізнесу. Ціна такої помилки може бути значно вища за вартість розробки і підтримки автотестів.

Вимірювання ефективності допомагає відповісти на питання: «Чи варто впроваджувати автоматизацію на проекті?», «Коли впровадження принесе нам значущий результат?», «Скільки годин ручного тестування ми відшкодується?»,

«Чи можна замінити 3 інженерів ручного тестування на 1 інженера автоматизованого тестування ?" та ін.

Для розрахунку ефективності впровадження автоматизації я використовую наступну загальну формулу розрахунку рентабельності інвестицій (ROI)[12]:

$$ROI = \frac{(Gain - Investments)}{Gain} \times 100\%, \quad (1)$$

де Gain – прибуток, розрахункова вартість тестування без автоматизації;

Investments – інвестиції, витрати на створення і виконання автоматичної бібліотеки тестів за той же період, що був використаний для обчислення розрахункової вартості тестування без автоматизації.

Якщо значення ROI має від'ємне значення, це значить, що клієнт через автоматизацію отримує збитки, що не є ефективним для впровадження у процес розробки. Якщо значення нульове – то від автоматизації немає прибутку та збитку, впровадження автоматизації не є ефективним, окрім випадків, коли розробка є довгостроковою. Якщо ROI більше 1, то клієнт отримує прибуток, у впровадженні автоматизації є сенс.

При розрахунку витрат на автоматизацію тестування, необхідно враховувати такі фактори:

- вартість програмного забезпечення для автоматизації тестування;
- витрати на навчання персоналу або залучення більш кваліфікованого персоналу;
- вартість розробки первісної бібліотеки автоматичних скриптів;
- вартість прогону бібліотеки автоматичних скриптів;
- витрати на аналіз результатів прогону автоматичних тестів;
- витрати на підтримку автоматизованих скриптів в актуальному стані.

Розрахувати очікувану вигоду від впровадження автоматичного тестування можна не тільки з боку матеріальних витрат, а також і з точки зору ефективності використання ресурсів. Даний метод заснований на порівнянні витрат часу, необхідних на впровадження, виконання, аналіз результатів і підтримка автоматичних тестів (Інвестиції) з витратами на ручні тести (Прибуток). Варто відзначити, що метод враховує тільки часові витрати персоналу, не беручи до уваги їх грошове вираження, тому він може бути особливо корисним у випадках, коли комерційні деталі невідомі[12].

Пропонується використовувати наступну формулу для розрахунку часових інвестицій в автоматизацію тестування за виділений період (TI_p)[12]:

$$TI_p = TI_0 + TC_0 + \sum_{n=1}^{k} (TC_e + TC_a + TC_m), \quad (2)$$

де TI_0 – стартові часові інвестиції, які в даному випадку складаються з часу витраченого на пошук відповідного програмного забезпечення для автоматизації тестування та навчання персоналу. Даний параметр може приймати нульове значення, наприклад, якщо залучалися вже навчені фахівці, і не стояло проблеми вибору програмного забезпечення для автоматизації тестування;

TC_0 – часові витрати на розробку початкової бібліотеки автоматичних скриптів, яка вираховується як середнє час, необхідне на написання одного скрипта (в годинах), помножене на загальну кількість скриптів;

k - кількість планованих циклів тестування (тобто передбачувана кількість прогонів скриптів);

TC_e – час, витрачений тестувальником на підготовку до виконання і безпосередньо на виконання одного скрипта (в годинах), помножене на загальну кількість скриптів. Дана змінна може приймати 0 значення, наприклад, коли

мається на увазі повністю автономне виконання тестів, що не вимагає втручання людини ні на стадії підготовки до виконання тесту, ні під час виконання тесту;

TC_a – часові витрати на аналіз результатів одноразового виконання набору автоматизованих скриптів, які обчислюються як передбачуваний відсоток невдало проведених тестів, помножений на загальну кількість скриптів і на середній час, необхідний на аналіз причин невдалого виконання одного скрипта (в годинах);

TC_m – часові витрати на підтримку автоматизованих скриптів в актуальному стані, які обчислюються як очікуваний коефіцієнт змін скриптів для кожного циклу виконання, помножене на середній час, потрібний на зміну одного скрипта одним тестувальником (в годинах), і на загальну кількість скриптів. Дана змінна може приймати 0 значення, наприклад, якщо в даній функціональній області не планується ніяких подальших змін.

Часові витрати на тестування за той же період, але без автоматизації (TG_p) можуть бути представлені в наступному вигляді[12]:

$$TG_p = TG_0 + \sum_{n=1}^{k} (TG_e + TG_a + TG_m), \quad (3)$$

де TG_0 – час, витрачений на розробку початкової бібліотеки ручних скриптів (в годинах). Дане значення може дорівнювати 0, якщо мається на увазі наявність існуючої бібліотеки скриптів;

k – кількість планованих циклів тестування (тобто передбачувана кількість прогонів скриптів);

TG_e – часові витрати на одноразове виконання набору ручних скриптів, які обчислюються як середнє час, витрачений на підготовку до виконання і безпосередньо на виконання одного скрипта (в годинах), помножене на загальну кількість скриптів;

TG_a – часові витрати на аналіз результатів виконання одного циклу набору ручних скриптів, які обчислюються як передбачуваний відсоток невдало проведених тестів, помножений на середній час, потрібний на аналіз причин невдалого виконання одного скрипта одним тестувальником (в годинах), і на загальну кількість скриптів. Дана змінна дорівнює 0 в більшості випадків, так як розглядаються ручні скрипти, які представляють детально описану інструкцію для тестувальника;

TG_m – часові витрати на підтримку ручних скриптів в актуальному стані, які обчислюється як очікуваний коефіцієнт змін, помножений на середній час, необхідний на зміну одного скрипта одним тестувальником (в годинах), і на загальну кількість скриптів. Дана змінна може приймати 0 значення, наприклад, якщо в даній функціональній області не планується ніяких подальших змін.

3.4 Опис методів розв’язання

Для розрахунку ефективності впровадження автоматизації я використовую формули (2) для розрахунку часових інвестицій в автоматизацію тестування за виділений період та формулу (3) часових витрат на тестування за той же період, але без автоматизації (TG_p).

Стандартні значення для початкових даних буде знайдено за допомогою методу декомпозиції для оцінки часу на виконання тесту. Для цього, необхідно знайти дію, яку можна прийняти за початкове мінімальне значення, для якого треба витратити певну кількість часу, та відштовхуючись від цього значення, оцінити інші дії та знайти загальну часову вартість одного тестового сценарію. Таким чином, за стандартне значення приймаємо 5 хвилин для введення значення у поле, згідно з такими розрахунками було підраховано, що кількість часу, яке необхідно витратити для кожного тестового сценарію, становить 20 хвилин середньо[13].

Для розрахунку часових інвестицій в автоматизацію тестування маємо такі початкові дані:

- загальна кількість тестів – 40;
- TI_0 – стартові часові інвестиції – 80 годин, так як багато часу було витрачено на пошук ефективних рішень для автоматизації тестування;
- TC_0 – середній час, необхідний на написання одного скрипта – 1 години, загальна кількість скриптів – 40, добуток поданих значень - 40;
- k – кількість планованих циклів тестування – 30, по одному разі кожного дня протягом одного місяця;
- TC_e – змінна має нульове значення, тому що даний проект повність автоматизує процес проходження тестів та не вимагає втручання людини на стадії підготовки до виконання тесту чи під час виконання тесту;
- TC_a – передбачуваний відсоток невдало проведених тестів – 20% (веб-портал не має складної функціональності, тести проходять швидко), загальну кількість тестів - 40, середній час, необхідний на аналіз причин невдалого виконання одного тесту – 0.5 години, добуток поданих значень дорівнює 4;
- TC_m – очікуваний коефіцієнт змін скриптів для кожного циклу виконання - 0.2 (стале середнє значення для кожного циклу), середній час, потрібний на зміну одного теста одним тестувальником – 0.25 години (може відрізнятись в залежності від кількості виправлень), на загальну кількість тестів – 40, добуток поданих значень - 2.

Використовуючи формулу маємо:

$$TI_p = 80 + 40 + \sum_{n=1}^{k}(0 + 4 + 2) = 300.$$

Для розрахунку часових інвестицій без автоматизації маємо такі початкові дані:

- загальна кількість тестів – 40;
- TG_0 – час, витрачений на розробку початкової бібліотеки ручних скриптів – 40 годин;
- k – кількість планованих циклів тестування - 30;
- TG_e – середній час, необхідний на підготовку до виконання і безпосередньо на виконання одного сценарію – 0.25 години, загальну кількість скриптів – 40, добуток поданих значень - 10;
- TG_a – дана змінна приймає нульове значення, так як розглядаються ручні скрипти, які представляють детально описану інструкцію для тестувальника;
- TG_m – очікуваний коефіцієнт змін – 0.2, середній час, необхідний на зміну одного скрипта одним тестувальником – 0.25 година, загальна кількість скриптів - 40. Добуток – 2.

Використовуючи формулу маємо:

$$TG_p = 40 + \sum_{n=1}^{k}(10 + 0 + 2) = 400.$$

Підставляючи отримані значення у загальну формулу ROI отримуємо:

$$ROI = \frac{(400 - 300)}{400} \times 100\% = 25.$$

Отже, для проекту впровадження автоматизації є прибутковим, так як отримане значення більше за одиницю.

Також варто враховувати той факт, що процес розподілу часових ресурсів у процесі автоматичного та ручного тестування відрізняється, про це свідчить рисунок 3.1:

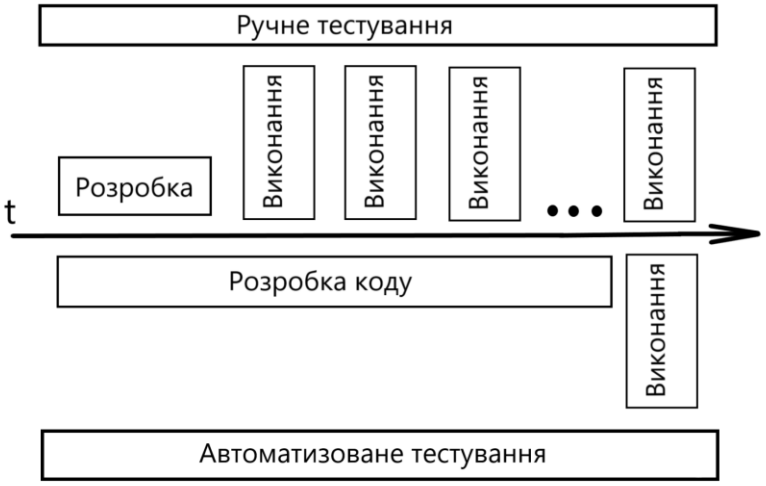


Рисунок 3.1 – Графік розподілу часових ресурсів

На початковій стадії введення автоматизованого тестування багато часу та зусиль витрачається на створення працюючого фреймворку та тестових наборів, цей факт також означає, що необхідно залучити кваліфікованих спеціалістів та відібрати певні інструменти для розробки системи. Але, вже після розробки та налагодження програмного забезпечення кількість часу та зусиль тестувальника набагато зменшується, адже зараз його робота полягає у аналізу пройдених тестів та, рідше, у розширенні тестових шаблонів та зміні вхідних даних.

Для ручного тестування процес створення тестових сценаріїв та тестових даних є набагато коротшим за процес налагодження початкового стану для можливості тестування. Тестувальник за ручним тестуванням витрачає весь свій час на виконання тестів вручну та створення статистичного звіту кожен раз (так як для даного проекту проходження тестів необхідно кожного дня, то цей процес буде проходити кожного дня).

Для того, щоб краще зрозуміти розподіл зусиль за двома варіантами тестувань, варто відобразити його на графіку, який представлено нижче на рисунку 3.2:

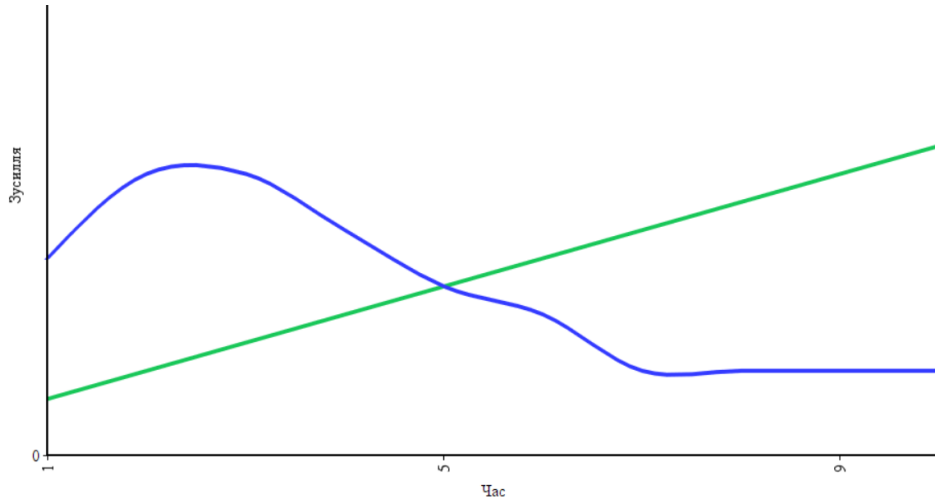


Рисунок 3.2 – Графік розподілу зусиль для обох видів тестувань

Очевидно, що автоматизація даних сценаріїв несе значний економічний ефект, який настає досить швидко і впевнено збільшується з часом.

Висновок до розділу

В даному розділі було проаналізовано методи визначення ефективності впровадження автоматизації у процес розробки програмного забезпечення шляхом використання методу підрахунку ROI. Для знаходження ROI було порівняно ефективність використання ручного тестування та автоматизації у вигляді часового виразу кожного з них. Отримане значення ROI означає, що автоматизація навіть на поточному рівні розробки програмного забезпечення є ефективним та принесе замовнику прибуток.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

4.1.1 Selenium

Selenium – інструмент автоматизованого управління браузерами. Найбільше поширення затосування Selenium спостерігається у сфері автоматизованого тестування веб-додатків. Також використовуючи Selenium можна добитися автоматизації будь-яких інших дій, що виконуються у браузері.

Розробку Selenium підтримують більшість розробників популярних браузерів. Іноді в популярних браузерах реалізована інтегрована підтримка Selenium. Selenium - центральний компонентом цілого ряду фреймворків автоматизації, а також інших інструментів автоматизації.

Selenium підтримує мобільні та десктопні браузери. Selenium допомагає розробці сценаріїв автоматизації з використання будь-якої мови програмування. Використовуючи Selenium можна створити розподілені стенди, які складатимуться з великої кількості машин, що матимуть різні браузери та операційні системи, і навіть виконувати хмарні сценарі.

Selenium WebDriver являється програмною бібліотекою по управлінню браузерами. Також вживається коротка назва – WebDriver.

WebDriver називають «драйвером браузера», але це не один драйвер для окремого браузера, а цілий ряд драйверів для окремих браузерів. WebDriver також включає набір клієнтських бібліотек, що виконані різними мовами програмування. Це основний продукт, проекту Selenium. Основні компоненти WebDriver представлені на рисунку 4.1:

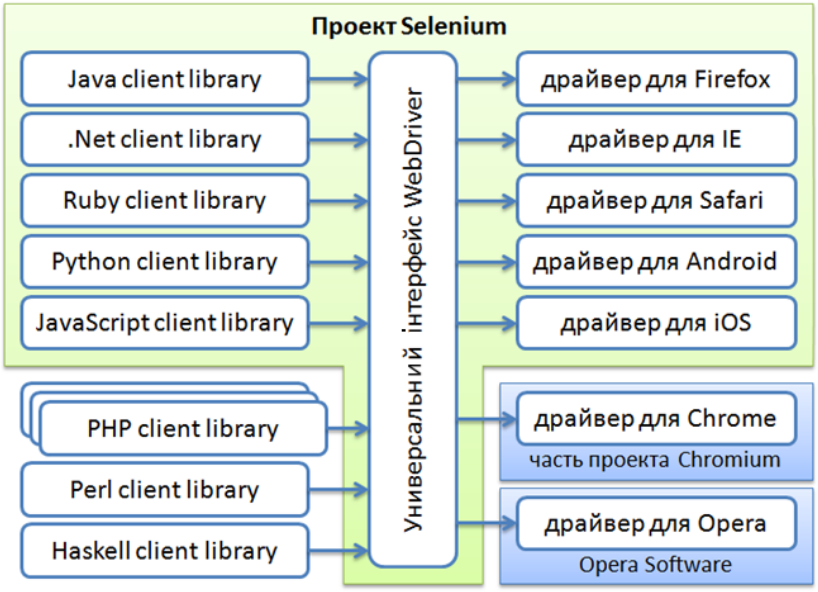


Рисунок 4.1 – Компонентні складові WebDriver

Selenium розробляє драйвери для таких браузерів: Firefox, Safari та Internet Explorer. Також Selenium розробляє драйвери для мобільних браузерів на системах Android та iOS. В рамках проекту Chromium розробляється драйвер для Google Chrome. В рамках проекту Opera Software розробляється драйвер для Opera (включаючи мобільні версії). Тому вони формально не вважаються часткою проекту Selenium, вони підтримуються та поширюються незалежно. Не зважаючи на це їх можна вважати належними до сімейства продуктів Selenium.

4.1.2 NUnit

NUnit – це фреймворк для тестування з відкритим вихідним кодом. Основне призначення даного фреймворку полягає у наданні можливості позначити тестові методи як тести. Вся основна функціональність це додавання атрибутів до тестів, окрім атрибуту [Test], який позначає метод як тест, NUnit має безліч інших атрибутів, за якими можна налагоджувати тестові методи, параметризувати їх та об’єднувати у тестові набори.

Атрибути, до яких NUnit надає доступ:

– Author Attribute – надає можливість визначити автора тесту;

- Category Attribute – визначає категорію (одну чи більше) тесту;
- Description Attribute – додає опис до тесту, тестового набору або збірки;
- Ignore Attribute – визначає тест, який не буде виконуватись;
- OneTimeSetUp Attribute – визначає метод, який буде виконуватись один раз для збірки тестів перед їх виконанням;
- OneTimeTearDown Attribute – визначає метод, який буде виконуватись один раз для збірки тестів після їх виконання;
- Order Attribute – визначає порядок виконання тестових методів;
- SetUp Attribute – визначає метод, який буде виконуватись для кожного тестового методу перед його виконанням;
- SetUpFixture – визначає метод, який буде виконуватись для кожного тестового набору перед їх виконанням;
- TearDown Attribute – визначає метод, який буде виконуватись для кожного тестового методу після його виконання;
- Test Attribute – атрибут, який позначає метод як тест;
- TestCase Attribute – атрибут позначає метод як тест з параметрами;
- TestCaseSource Attribute – надає можливість додати до тесту ресурс, звідки будут обрані параметри;
- TestFixture Attribute – помічає клас як тестовий набір;
- TestFixtureTearDown Attribute – визначає метод, який буде виконуватись для кожного тестового набору після його виконання.

Також NUnit надає доступ до бібліотеки класів та методів, основа функція яких полягає у перевірці актуальних даних до очікуваних.

Assertion клас є основою для модульного тестування в будь-який з платформ xUnit, і NUnit не є винятком. NUnit надає багатий набір тверджень у вигляді статичних методів класу Assert.

Якщо твердження не виконано, виклик методу не повертається і видається повідомлення про помилку. Якщо тест містить кілька тверджень, то будь-який, наступне за помилковим, не буде виконано. З цієї причини зазвичай краще спробувати одне твердження на тест.

Кожен метод може бути викликаний без повідомлення, з простим текстовим повідомленням або з повідомленням і аргументами. В останньому випадку повідомлення форматується з використанням наданого тексту і аргументів[14].

Окрім цього програмне забезпечення NUnit надає можливість оперувати тестовими даними, використовуючи атрибути, за якими програма буде генерувати параметри за певним атрибутом:

- Sequential Attribute – використовується у тесті для вказівки, що NUnit повинен генерувати тестові випадки, обираючи окремі елементи даних, надані для параметрів тесту, без створення додаткових комбінацій. Якщо дані параметрів надаються декількома атрибутами, порядок, у якому NUnit використовує елементи даних, не гарантується. Тим не менш, можна очікувати, що вона залишатиметься постійною для даного середовища виконання. Бажано Sequential Attribute використовувати тільки один атрибут даних для кожного параметру[15];
- Combinatorial Attribute - використовується у тесті, щоб вказати, що NUnit повинен генерувати тести для всіх можливих комбінацій окремих елементів даних, що надаються для параметрів тесту. Оскільки це є параметром за замовчуванням, використання цього атрибута є необов'язковим [15];
- Pairwise Attribute – використовується у тесті, щоб вказати, що NUnit повинен формувати такі набори даних, в яких кожне

тестоване значення кожного з перевірених параметрів хоча б один раз поєднується з кожним тестованим значенням всіх інших параметрів, що перевіряються. Це добре відомий підхід лдя скорочення параметрів, на відміну від комбінаторного методу, коли задіяні більше двох ознак (параметрів). Наприклад, використовуючи атрибут Combinatorial, наступний тест (рисунок 4.3) буде виконано 12 (3x2x2) разів. За допомогою атрибуту Pairwise вона виконується лише достатньо раз, щоб покрити кожну можливу пару (рисунок 4.4)[15];

```
[Test, Pairwise]
public void MyTest(
    [Values("a", "b", "c")] string a,
    [Values("+", "-")] string b,
    [Values("x", "y")] string c)
{
    Console.WriteLine("{0} {1} {2}", a, b, c);
}
```

Рисунок 4.3 – Тест для проведення pairwise вибірки

```
a + y
a - x
b - y
b + x
c - x
c + y
```

Рисунок 4.4 – Результати параметрів, з якими проведеться тест

4.1.3 TeamCity

Програмне забезпечення JetBrains TeamCity – це інтегроване середовище колективної розробки і система управління процесом створення програмного

забезпечення. TeamCity автоматизує і координує основні процеси спільної розробки, включаючи запуск тестів, аналіз вихідного коду, інтеграцію і надбудову кодових параметрів, заданих командою розробників, управління збірками. TeamCity прискорює процес розробки програмного забезпечення і гарантує ефективну взаємодію між членами команди.

Продукт TeamCity забезпечує інтеграцію і контроль модифікацій коду в процесі спільної роботи і автоматизує процес запуск тестів. Рішення контролює комплексну роботу тестів і виправлення помилок і здійснює генерацію оповіщень. Завдяки потужній функціональності, TeamCity дозволяє ефективно управляти розробкою і надає можливість використання всіх можливих типів конфігурації для будь-яких проектів.

Цей інструмент дозволяє:

– виконувати збірки паралельно на різних платформах і середовищах;
– оптимізувати процес інтеграції коду і бути впевненим у відсутності не працюючого коду у репозиторії;
– паралельно з виконанням переглядати результати виконання тестів.

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Для коректної роботи фреймворку необхідно використовувати певне програмне забезпечення та ресурсні бібліотеки для доступу до необхідних компонентів та методів. Мій проект виконаний на мові C# з використанням бібліотеки плагінів для запуску та повноцінного функціонування програм написаних в стандарті «.NET» .Net Framework 4.6 фреймворку для програмного засобу Visual Studio 2017 Community Edition. Такий набір початкового набору програм біло обрано через їх відкритий доступ, стабільність та швидкість

роботи і використання програмних забезпечень та ресурсних бібліотек, з якими я маю досвід роботи.

Для початку роботи необхідно дотриматись вимог до технічного забезпечення:

- процесор з тактовою частотою не нижче 1,8 ГГц. Рекомендується використовувати як мінімум двоядерний процесор;
- 2 ГБ оперативної пам'яті; рекомендується 4 ГБ оперативної пам'яті (мінімум 2,5 ГБ при виконанні на віртуальній машині);
- місце на жорсткому диску: до 120 ГБ вільного місця в залежності від встановлених компонентів, зазвичай для установки потрібно від 20 до 50 ГБ вільного місця;
- швидкість жорсткого диска: для підвищення продуктивності встановіть Windows і Visual Studio на твердотільний накопичувач (SSD);
- відеоадаптер з мінімальним дозволом 720p (1280 на 720 пікселів); для оптимальної роботи Visual Studio рекомендується дозвіл WXGA (1366 на 768 пікселів) або вище.

Доступ до бібліотек та компонентів використовується система управління пакетами NuGet, дана система надає доступ до відкритих бібліотек проектів для платформ розробки Microsoft для бібліотек .Net Framework. Для роботи фреймворку використовуються наступні пакети Nuget (важливо зазначити, що для початку роботи з фреймворком немає необхідності спеціалісту власноруч встановлювати пакети, адже проект вже містить необхідні дані та при першому білді проекту Visual Studio автоматично встановить їх до системи):

- JDI Core – надає доступ до загальних інтерфейсів та компонентів;
- JDI UIWeb – надає доступ до веб елементам та дозволяє позначати класи та поля як веб сторінка чи елемент;

- Log4net – пакет для створення логування дій;
- NUnit – загальна біліотека NUnit, містить атрибути, клас для перевірок тощо;
- NUnit Console та NUnit Console Runner використовуються для можливості запуску тестів;
- Selenium Web driver – надає доступ до веб драйверу та методам його управління;
- Selenium Support – розширені методи для роботи з веб драйвером;
- System ValueTuple – системний клас для створення кортежей;
- WebDriver Manger – керування веб драйвером.

4.3 Архітектура програмного забезпечення

4.3.1 Діаграма класів

Загально структура класів поділена на інфраструктурні рівні автоматизації: Core, Context, Step, Test. Таке розподілення дозволяє інкапсулювати використання певних елементів та методів для запобігання непотрібних зв’язків між проектами.

На рівні Core проекту розташовані необхідні класи та компоненти для загальної логіки роботи з драйвером, опис роботи з веб елементами, притаманні для веб-порталу, що тестується. Для роботи з вхідними файлами використовується extension клас XMLExtensions, такий підхід дозволяє десеріалізувати xml файл на рівні Test у класі TestExtendedData. Окрім цього, у даному проекті описано контекстний клас та методи створення сесій для тесту. Також описані моделі об’єктів, з якими тестується робота веб-порталу.

Рівень Context розмежується на два підрівні: Page та Context. У першому описуються об’єкти веб сторінок та веб елементів шляхом написання локаторів та використанні відповідних атрибутів NUnit та JDI Framework. Процес

ініціалізації цих елементів здійснюється автоматично кожного разу, коли викликається певний елемент (це є перевагою використання патерну PageFactory) у методах Context. На підрівні Context описується бізнес-логіка використання веб елементів та створення загальних методів використання елементів та форм. Класи контексту створюються за їх логічним використанням веб елементів тих чи інших веб сторінок. Варто зазначити, що для поточного рівня мого проекту немає сенсу розподіляти Page та Context на різні підрівні, так як кількість функціоналу веб-порталу обмежена. Цей рівень має extension клас ITPortalContext з загальними методами, які можуть використовуватись у будь-який момент тесту, не прив'язуючись до сторінок.

На рівні Step створюються методи, так звані «степи» для подальшого використання їх у тестах. Логічно діляться на класи з назвою використовуваних контекстів, проте дозволяється створювати загальні степи для рутинних дій (логін, перевірка полів, тощо).

На рівні Test створюються класи з наборами тестів, які тестують певну функціональність веб-порталу. Також створено теку для зберігання вхідних даних та клас для серіалізації та десеріалізації xml файлів. На цьому рівні описано початкові дії перед кожним стартом тесту та їх закінчення.

Схема класів представлена на рисунку 4.5.

Рисунок 4.5 – Діаграма класів

4.3.2 Діаграма послідовності

Коли тест запускається на виконання, перш за все, виконуються початкові дії для приготування всіх даних для роботи. Зі зверненням до класу

BaseTestFixture генерується унікальна сесія, прив'язана до процесу проходження конкретного тесту. Після цього ініціалізуються статичні об'єкти вхідних даних, подані у класі TestExtendedData. Наступне звернення до класу Hooks створює об'єкт для проходження тесту NUnit runner, закриває усі браузери, які могли бути відкриті, ініціалізує веб сайт та відкриває браузер на початковій сторінці. Після цього виконуються методи тесту. Кожен метод, описаний у тесті, викликає метод Step, який використовує методи Context, де методи бізнес логіки звертаються да Page та створююсь об'єкти сторінок та елементів. Така схема існує для кожного виклику методу Step із тесту, після чого виконується перевірка за повернутими даними з тими, що ми очікуємо. У кінці браузер закривається та видаляються дані про поточну сесію.

Схема послідовності представлена на рисунку 4.6.

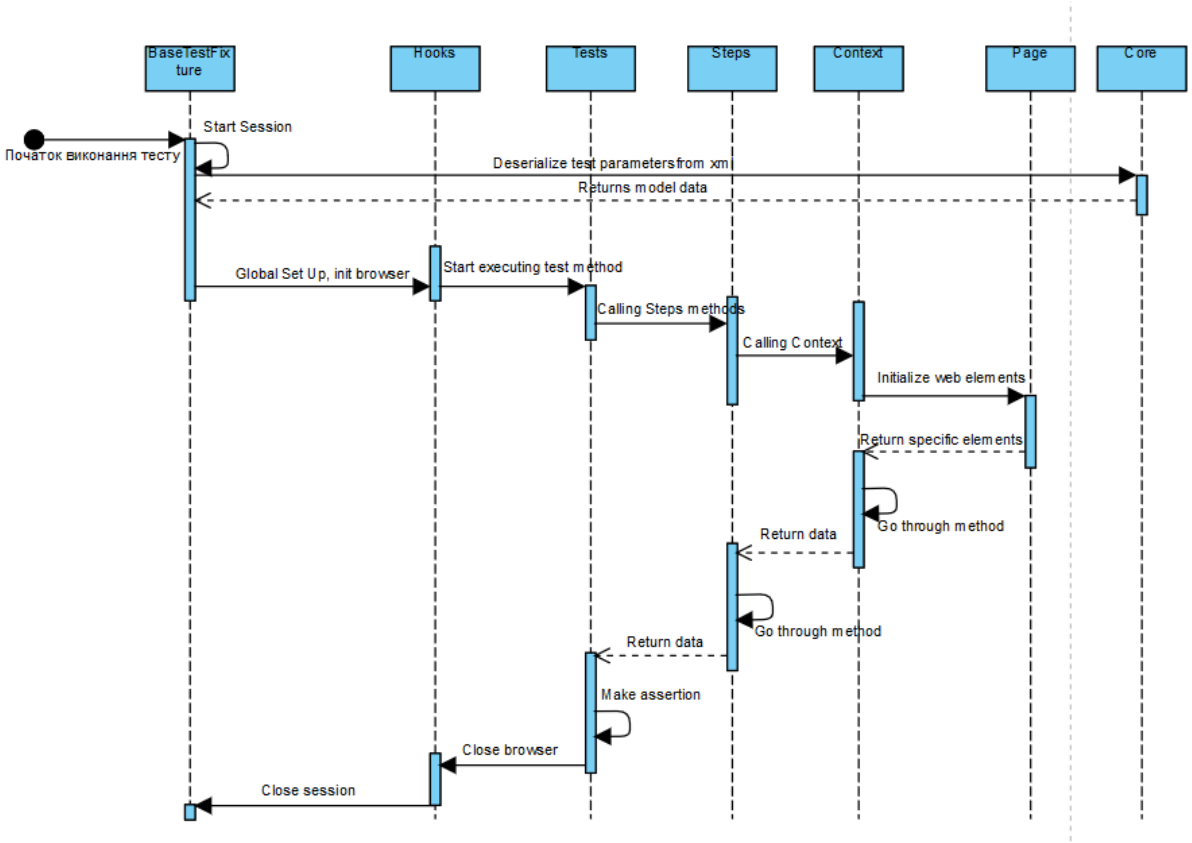


Рисунок 4.6 – Діаграма послідовності

4.3.3 Діаграма компонентів

Усього у фреймворку існує 3 компоненти, які представляють загальну структуру взаємозв'язку фреймворку та систем, з якими існує інтеграція. У фремворку розташовано 4 компоненти, які являють собою інфраструктурні рівні автоматизації. Кожен з них відповідає за певну функціональність та повертає певні дані.

Компонент Core зберігає основні функції щодо ініціалізації елементів та методів генерації об'єктів. Компонент Context відповідає за створення об'єктів веб елементів та методів звернення до них. Компонент Step зберігає методи комунікації з контекстними методами обробки веб елементів та надає доступ до їх використання для тестів. Компонент Test виконує роль запуску теста, оперуючи методами Step та вхідними параметрами.

Компонент Git – це система управління версіями сервісу Bitbucket. Відповідає за віддалене зберігання головної гілки репозиторію та відгалужених гілок. Дозволяє створювати запити на об'єднання гілок та керувати процесом розробки програмного коду та рівнем доступу до змін.

Кмпонент TeamCity дозволяє виконувати білд репозиторію та тестові сценарії автоматично на віддаленому агенті, зв'язок агента та репозиторію здійснюється за рахунок його з'єднання з віддаленим репозиторієм Git.

Схема компонентів представлена на рисунку 4.7. Детальна діаграма компонентів представлена у графічному матеріалі.

Рисунок 4.7 – Діаграма компонентів

4.3.4 Специфікація функцій

У таблиці 4.1 представлено список основних функцій створеного фреймворку у вигляді методів та їх опису.

Таблиця 4.1 – Перелік основних функцій фреймворку

Функція чи метод	Опис
KillAllRunWebDrivers()	Вбиває будь які існуючі процеси браузеру, з яким відбувається взаємодія у тесті
Open()	Відкриває браузер
StartSession()	Зчитує id сесії, згенерованої NUnit та додає до контексту
CloseSession()	Видаляє id сесії із контексту
FromXML<T>	Десеріалізує xml файл у модель, подану у тип
ToXML<T>	Серіалізує модель, подану у тип у xml файл
Current	Зберігає дані конкретного тесту в контексті у вигляді статичного TestrunContext
IsLoginPageOpened()	Перевірка чи відкрита логін сторінка, повертає bool
GetPageHeader()	Повертає титул сторінки
[FindBy(XPath =	Поле об’явлення форми регістрації та

".//div[@class='registration-login-form']")] public RegistrationLoginForm RegistrationLoginForm;	логіну за допомогою атрибута FindBy та локатору xpath, об'єкт зберігається з типом RegistrationLoginForm, який у свою чергу також зберігає веб елементи
GetControlPanelButton(string buttonName)	Повертає кнопку з панелі управління на сторінці логіну за її назвою
GetErrorMessageFor (string type)	Повертає текст помилки, якщо така присутня на певній формі
SelectForm(string form)	Знаходить за назвою форму та обирає її на сторінці, якщо така не обрана
IsErrorPresent()	Повертає bool значення чи присутнє повідомлення про помилку на сторінці
SetUserData(User user)	Заповнює поля інформацією користувача
SubmitUser(User user)	Заповнює поля інформацією користувача та подає зміни до системи
public List<Label> EmailErrorLabels	Приклад оголошення колекції веб елементів
LoginToPortalAs(User user)	Логін до системи як користувач, зазначений у параметрі
LoginToPortalAsDefaultUser()	Логін до системи як користувач за замовченням
Assert.That(expected, equal, message)	Перевірка чи збігається фактичний результат з очікуваним, якщо ні – повертає помилку с текстом, зазначеним у параметрі

4.4 Опис звітів

Користувач отримує два види звітів: про виконання тестів та результати проходження білду – у вигляді повідомлення на поштову скриньку. У звіті з результатами виконання тестів є такі поля:

– назва гілки, з якої було виконано проходження тестів;

– статус (чи успішне загальне виконання тестів: якщо хоч один тест не пройшов, статус failed);

– перелік змін, які були зроблені у гілці на момент початку проходження тестів;
– кількість пройдених тестів;
– кількість тестів, що не пройшли;
– причина, за якою тести не пройшли у вигляді логованих даних.

У звіті з результатами виконання білду є такі поля:

– назва гілки, для якої був виконаний білд;
– статус (чи успішне виконання білду);
– перелік комітів, які були додані до гілки.

Загальний вигляд звіту про виконання білду представлено на рисунку 4.8.

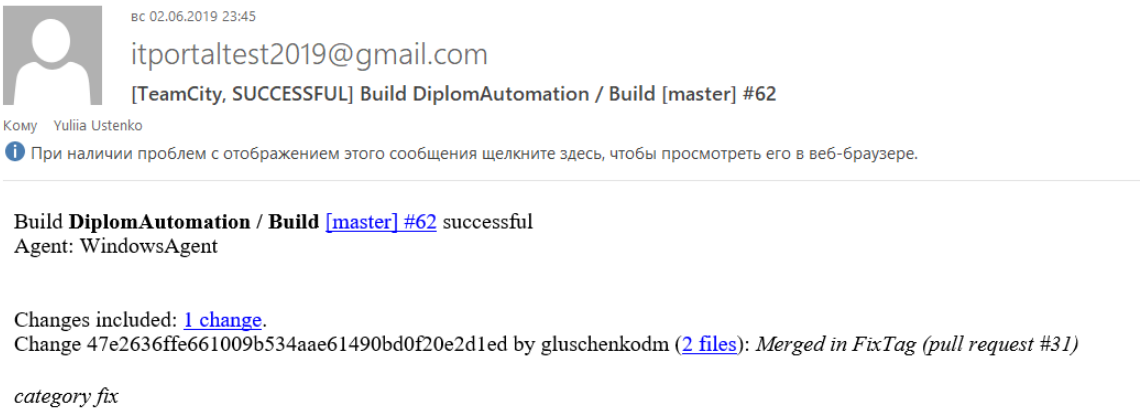


Рисунок 4.8 – Звіт про виконання білду

Загальний вигляд звіту при проходженні тестів представлено на рисунку 4.9.

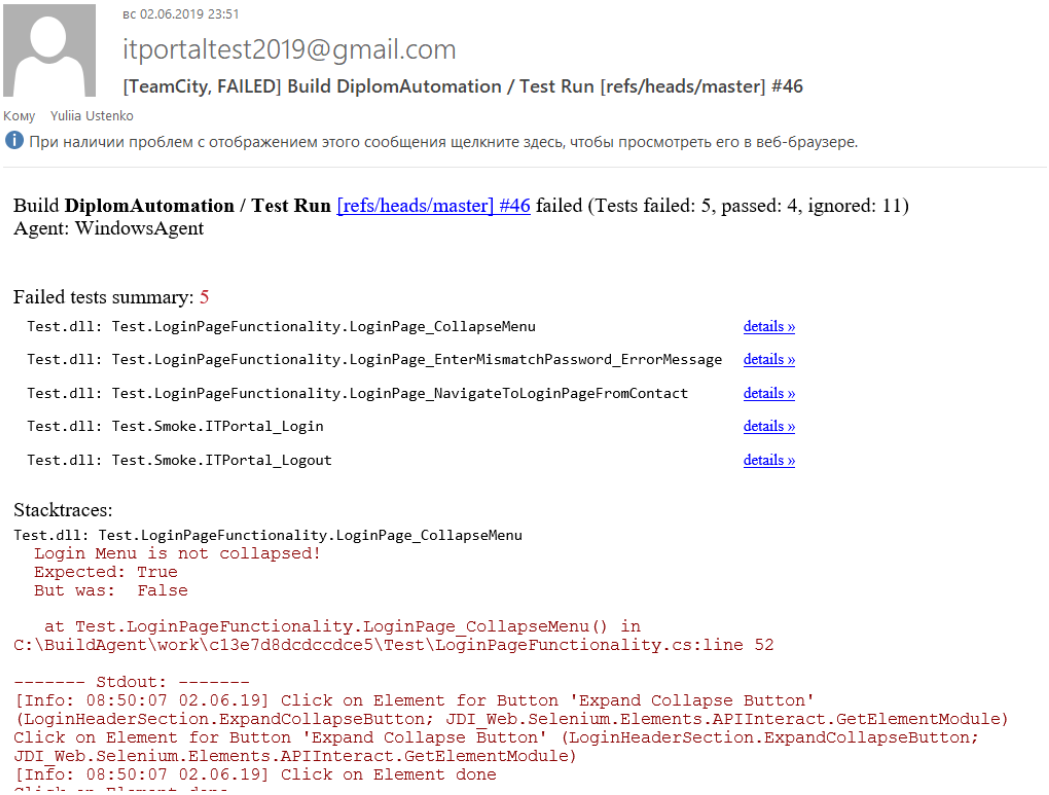


Рисунок 4.9 – Звіт про проходження тестів

Зміст звіту за тестом, що не пройшов скаладється з таких пунктів:

– фактичний шлях тесту;
– загальна причина, за якою тест не пройшов (у більшості випадків, повідомлення методу Assert());
– метод, на якому було викликано повідомлення про помилку;
– покрокове проходження тесту по методам та полям з описом шляху методу чи поля, логований текст кроку та час.

Висновок до розділу

В даному розділі були описані засоби, які використовуються для розробки проекту, як розробки фреймворку, так і інтеграції с засобами для спрощення процесу автоматизації, було проаналізовано їх переваги та шлях до використання у межах проекту; були описані загальні вимоги та розглянуто

діаграми: класів, яка зображує зв’язок між класами, послідовності, яка описує порядок виклику класів та методів та компонентів, яка зображує компоненти та їх зв’язок між собою. Також було позначено основні функції фреймворку та їх опис та розглянуто модель звітності за результатами.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

У процесі розробки основна увага приділялась чотирьом основним функціям проекту, а саме:

- автоматизація тестів. Сенс даного пункту є створення гнучкого фреймворку, за яким спеціалісту буде зручно орієнтуватись та розширювати функціонал та створювати нові тести та тестові набори;
- система параметризації. Особливо важливо було створити функціональний набір, мета якого оперувати вхідними даними у вигляді файлу з xml-розміткою та конвертувати його у модель;
- інтеграція з системою керування версіями Git. Це дозволяє створювати віддалений доступ до коду та надає можливість роботи над кодом декільком спеціалістам одразу.
- інтеграція з системою неперервної інтеграції TeamCity, що дозволяє виконувати тести віддалено та без участі людини.

5.1.1 Автоматизація тестів

Тестувальник має можливість у рамках фреймворку створювати тестові сценарії та тестові дані. Для того, щоб створити новий тестовий набір або фічу, необхідно створити новий файл класу у проекті Test. Приклад тестового набору та тестів зображено на рисунку 5.1.

```
[TestFixture]
[Category("Smoke")]
0 references | Юлия Устенко, 4 days ago | 4 authors, 11 changes | 4 work items
public class Smoke : BaseTestFixture
{
    private readonly CommonSteps _loginSteps = new CommonSteps();
    private readonly NewsAndProfileSteps _newsAndProfileSteps = new NewsAndProfileSteps();

    [Test]
    0 references | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public void ITPortal_Login()
    {
        var expectedHeader = "Стрічка новин";

        _loginSteps.LoginToPortalAsDefaultUser();
        var actualHeader = ITPortalContext.IsNewsPageOpen();

        Assert.That(actualHeader, Is.EqualTo(expectedHeader.ToUpperInvariant()), "The News Page isn't opened!");
    }

    [Test]
    0 references | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public void ITPortal_Logout()
    {
        _loginSteps.LoginToPortalAsDefaultUser();
        _newsAndProfileSteps.LogoutFromPortal();

        var actualIsOpened = ITPortalContext.IsLoginPageOpened();

        Assert.That(actualIsOpened, "The Login Page isn't opened!");
    }
}
```

Рисунок 5.1 – Тестовий набір

Для того, щоб оголосити клас як тестовий набір необхідно додати атрибут [TextFixture], як показано на рисунку 5.2.

```
[TestFixture]
0 references | 0 changes | 0 authors, 0 changes
public class NewTests
{

}
```

Рисунок 5.2 – Створення тестового набору

Для того, щоб оголосити метод у тестовому класі як тест, необхідно додати атрибут [Test], як показано на рисунку 5.3.

```
[TestFixture]
0 references | 0 changes | 0 authors, 0 changes
public class NewTests
{
    [Test]
    0 references | 0 changes | 0 authors, 0 changes
    public void Test_Something()
    {

    }

}
```

Рисунок 5.3 – Додавання тесту до тестового набору

Для створення нового веб елементу або його зміни, необхідно звернутися до проекту Context тек Page, Form або Section. Наприклад, зміст логін сторінки у фреймфорку виглядає, як на рисунку 5.4.

```
1 reference | Юлия Устенко, 4 days ago | 2 authors, 6 changes | 2 work items
public class LoginPage : WebPage
{
    [FindBy(XPath = ".//div[@class='registration-login-form']")]
    public RegistrationLoginForm RegistrationLoginForm;

    [FindBy(XPath = "//a[contains(text(),'Про проект')]")]
    public Button AboutProjectLink;

    [FindBy(XPath = ".//div[@class='header--standard-wrap']")]
    public LoginHeaderSection HeaderSection;

    [FindBy(XPath = ".//input[@type='submit']")]
    [Name("Submit")]
    public Button SubmitButton;

    1 reference | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public Button GetControlPanelButton(string buttonName)...

    1 reference | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public string GetErrorMessageForRegistration(string type)...

    1 reference | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public string GetErrorMessageForLogin(string type)...

    1 reference | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public void ClickOnExpandCollapseButton()...

    1 reference | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public bool IsLoginMenuDisplayed()...

    1 reference | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public void ClickOnGoToLoginPage()...

    1 reference | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public void SetTextToRegisterField(string fieldName, string text)...
}
```

Рисунок 5.4 – Зміст логін сторінки

Тут присутні елементи, які існують лише в рамках існування логін сторінки у браузері, наприклад, елемент форми для реєстрації та логіну виглядає, як на рисунку 5.5.

```
1 reference | Юлия Устенко, 2 days ago | 1 author, 2 changes | 2 work items
public class RegistrationLoginForm : Form
{
    [FindBy(XPath = ".//form[@action='/Account/Login']")]
    public LoginForm LoginForm;

    [FindBy(XPath = ".//form[@action='/Account/Register']")]
    public RegisterForm RegisterForm;

    [FindBy(XPath = ".//a[contains(text(),'Реєстрація')]")]
    [Name("Register")]
    public Button RegisterTab;

    [FindBy(XPath = ".//a[contains(text(),'Вхід')]")]
    [Name("Login")]
    public Button LoginTab;

    2 references | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public void SelectForm(string form)
    {
        var button = GetElementClass.GetButton(form);
        if (!button.GetAttribute( name: "class").Contains( value: "active"))
            button.Click();
    }
}
```

Рисунок 5.5 – Зміст форми для логіну чи реєстрації

Ця форма містить у собі два важливих елементи: вкладки логіну та реєстрації. Для керування цими елементами було створено метод, який натискає на вкладку, якщо необхідно відкрити якусь необхідну.

Форма логіну вигладяє, як на рисунку 5.6.

```
1 reference | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
public class LoginForm : Form<User>
{
    [FindBy(XPath = ".//input[@type='email']")]
    [Name("Email")]
    public TextField EmailField;

    [FindBy(XPath = ".//input[@type='password']")]
    [Name("Password")]
    public TextField PasswordField;

    [FindBy(XPath = ".//input[@type='submit']")]
    [Name("Submit")]
    public Button SubmitButton;

    1 reference | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public List<Label> EmailErrorLabels...

    2 references | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public string IsErrorPresent()...

    1 reference | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public void SetUserData(User user)...
}
```

Рисунок 5.6 – Зміст логін форми

Як видно з рисунків 5.4-5.6, у кожному веб класі створено методи для взаємодії з елементами, це методи Context рівня, які називаються контекстними методами.

Для створення нового веб елементу, необхідно створити нове поле з типом веб елементу та додати до нього атрибути для пошуку цього елементу за локатором. Додатково, можна присвоїти цьому об’єкту назву.

На рівні Steps можна створювати методи, які взаємодіють з веб елементами шляхом використання контекстних методів. Взаємодію цього проекту та контексту можна побачити у використанні простору імен Context, рисунок 5.7.

```
namespace Steps
{
    using Context;

    4 references | Юлия Устенко, 4 days ago | 1 author, 2 changes | 2 work items
    public class CommonSteps
    {
        2 references | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
        public void LoginToPortalAs(User user)
        {
            var loginPage = WebPortal.LoginPage.RegistrationLoginForm;
            loginPage.SelectForm("Login");
            loginPage.LoginForm.SetUserData(user);
            SessionService.Current.CurrentUser = user;
        }
```

Рисунок 5.7 – Взаємодія контекстного прошарку та Steps

5.1.2 Система параметризації

Для створення та керування існуючими вхідними параметрами тестувальнику необхідно, перш за все, створити файл з розширенням .xml та наповнити його даними у вигляді xml-розмітки та логічним розділенням даних згідно з моделлю об’єкту. Приклад створеного файлу зображено на рисунку 5.8.

Рисунок 5.8 – Приклад файлу з вхідними даними

Для десеріалізції файлу з вхідними параметрами наобхідно у класі TestExtendedData створити новий об’єкт(будь який об’єкт, реалізуючий IEnumerable), у якому необхідно викликати метод FromXML() та в параметрі вказати назву файлу. Приклад зображено на рисунку 5.9.

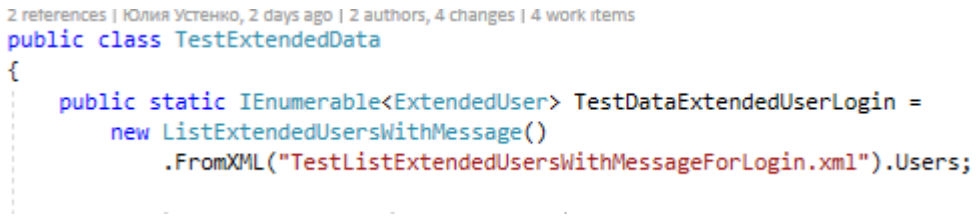


Рисунок 5.9 – Приклад створення об’єкту з вхідними даними

Теперь тестувальник має можливість використати створений об’єкт у тестовому методі, написавши атрибут [TestCaseSource], як зображено на рисунку 5.10.

```
[Test]
[TestCaseSource(typeof(TestExtendedData), "TestCasesLogin")]
0 references | Юлия Устенко, 3 days ago | 1 author, 3 changes | 3 work items
public void LoginForm_EnterWrongCredentials(ExtendedUser user)
{
    _loginSteps.LoginToPortalAs(user);
    var actualMessage = _loginSteps.GetErrorMessageForFielsInForm(field: user.Object.Item1, form: "Login");
    var expectedMessage = user.Object.Item2;

    Assert.That(actualMessage, Is.EqualTo(expectedMessage), "Actual message does not equal to expected!");
}
```

Рисунок 5.10 – Використання об’єкту з вхідними параметрами у тесті

5.1.3 Інтеграція з системою керування версіями Git

Тестувальник має можливість створювати відгалудженні гілки та локально працювати з кодом, але зберігати усі дані віддалено. Для того, щоб створити нову гілку, необхідно у сервісі Bitbucked у ролі Admin перейти до вкладки «Branches» та натиснути кнопку «Create branch». Як зображено на рисунку 5.11.

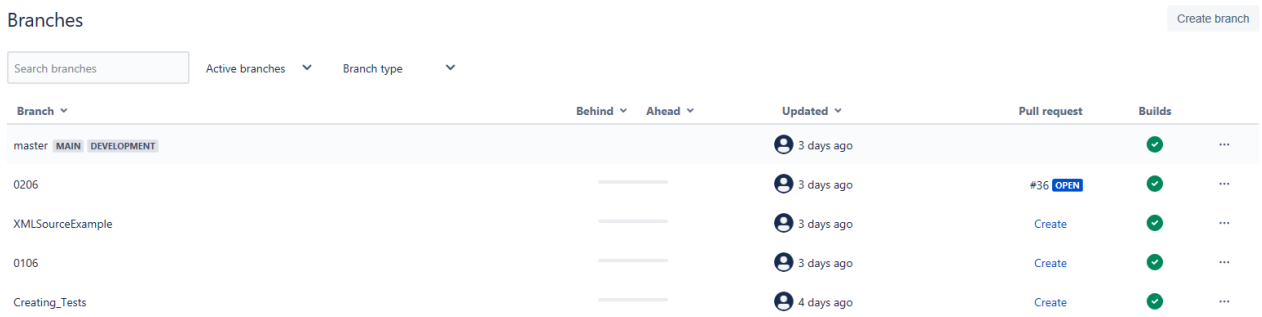


Рисунок 5.11 – Створення нової гілки

Для того, щоб відправити нові зміни до віддалної гілки або зібрати локально зміни, треба у Visual Studio у вкладці Team Explorer обрати пункт Changes, закомітити усі зміни, додавши назву та натиснути кнопку «Commit all and push», як зображено на рисунку 5.12.

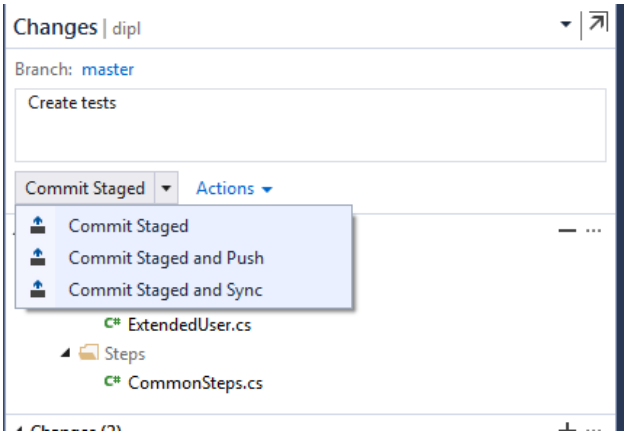


Рисунок 5.12 – Відправка змін до віддаленого репозиторію

Для того, щоб маги змогу з'єднати нові зміни з головною гілкою, треба створити запит на об'єднання та отримати схвалення одного з тестувальників. Запит створюється у сервісі Bitbucked. Необхідно обрати вкладку Pull request та натиснути кнопку Create pull request. Далі необхідно написати назву запиту та приєднати гілку, яку необхідно об'єднати, цей процес зображено на рисунку 5.13.

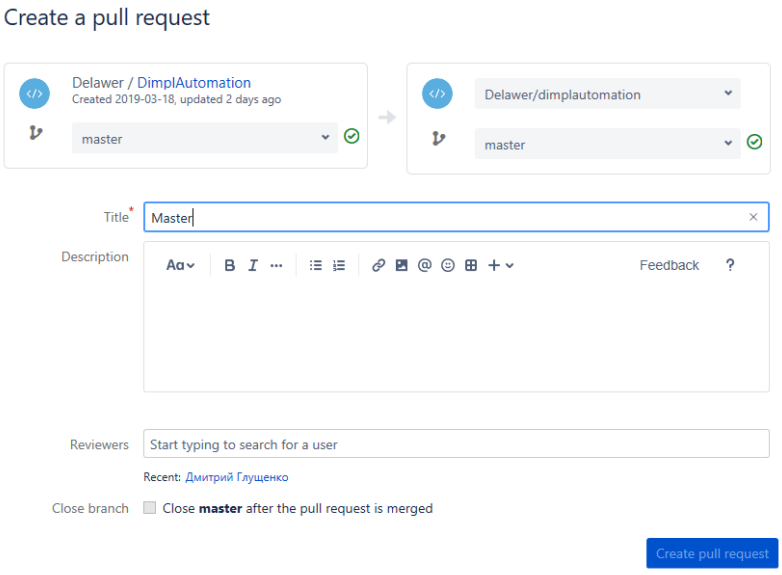


Рисунок 5.13 – Створення запиту

Для схвалення об’єднання необхідно у ролі Admin натиснути у запиті кнопку Approve, вже після цього можно з’єднати гілки, натиснувши кнопку Approve, знаходження цих кнопок зображено на рисунку 5.14.

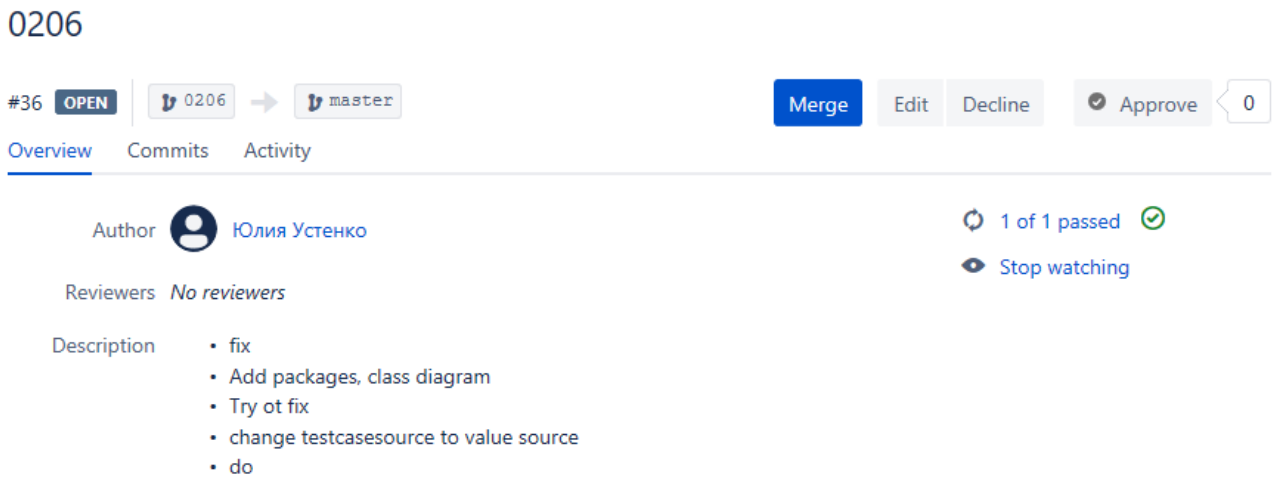


Рисунок 5.14 – Розміщення основних кнопок запиту

5.1.4 Інтеграція з системою неперервної інтеграції TeamCity

Зараз тестувальнику не треба докладати зусиль для того, щоб тести виконувались. У сервісі TeamCity цей процес проходить автоматично, тестувальник отримує кожного разу повідомлення про результат проходження на поштову скриньку.

Для тих випадків, коли тестувальнику необхідно додатково запустити виконання тестів, йому необхідно написнути на кнопку Run у джобі (обсяг ранів чи білдів) Test Run, як зображено на рисунку 5.15.

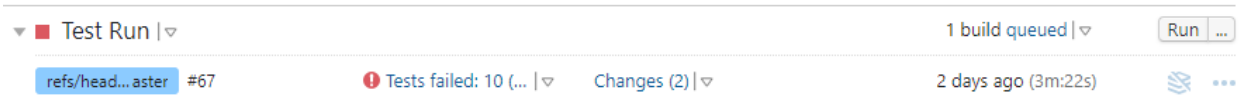


Рисунок 5.15 – Запуск виконання тестів вручну

Після цього запуск стає у чергу виконань, та виконується автоматично.

Кожного разу, коли тестувальник створює новий запит на об’єднання, у сервісі TeamCity автоматично проходить білд проекту та надсилається повідомлення про результат на поштову скриньку команди. Загальний вигляд таких білдів зображено на рисунку 5.16.

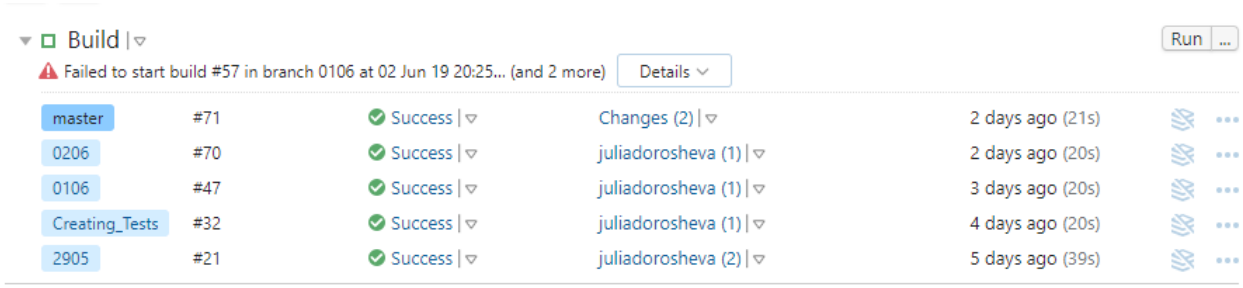


Рисунок 5.16 – Джоба для білду гілок

5.2 Випробування програмного продукту

Для випробування проекту буде перевірено такі функції:

- пошук веб елементу на сторінці;
- проходження тесту;
- перевірка виклику методів та дотримання рівнів автоматизації;
- відправлення локальних змін до віддаленого репозиторію;
- створення запиту на об’єднання;
- перевірка відображення результату білду на Bitbucket;
- запуск тестів;
- нотифікація тестувальника.

5.2.1 Мета випробувань

Метою випробувань є перевірка відповідності функцій комплексу задач системи автоматизації тестування веб-порталу вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

- ГОСТ 34.603-92. Інформаційна технологія. Види випробувань автоматизованих систем;
- ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

Результати випробувань наведені у таблиці 5.1

Таблиця 5.1 – Результати випробувань

Ситуація	Очікуваний результат
Пошук веб елементу під час виконання тесту	Веб елемент знайдений та з ним можна виконувати дії
Виклик методів згідно рівням	Методи викликаються та виконуються послідовно
Відправка локальних змін до віддаленого репозиторію через Visual Studio	Дані відравляються та відображаються у Bitbucket Git
Створення запиту на об’єднання коду та перевірка білду гілки	Гілка збілдилась та результати відображаються у запиті
Налагоджене автоматичне виконання тестів у TeamCity	Тести виконуються на агенті
Білд гілки та отримання нотифікації	Тестувальник отримав лист на поштову скриньку з результатами білду
Проходження тестів та отримання нотифікації	Тестувальник отримав лист на поштову скриньку зі звітом про виконання тестів

Висновок до розділу

У даному розділі було створено детальну інструкцію користувача для роботи з фреймворком та додатковими системами для автоматизації та управління. Були наведені скріншоти послідовності дій тестувальника для різних варіантів використань. Був виконаний набір тестів, таких як:

- пошук веб елементу на сторінці;
- проходження тесту;
- перевірка виклику методів та дотримання рівнів автоматизації;
- відправлення локальних змін до віддаленого репозиторію;
- створення запиту на об’єднання;
- перевірка відображення результату білду на Bitbucket;
- запуск тестів;
- нотифікація тестувальника.

ЗАГАЛЬНІ ВИСНОВКИ

У даній дипломній роботі було розглянуто та запропоновано спосіб зменшення часу, необхідного для виконання та аналізу тестів та виконано усі поставлені завдання, а саме:

- проведено аналіз існуючих аналогів та знайдено переваги та недоліки кожного з них;
- проведено аналіз існуючих засобів, які допомогають вирішити поставлену мету;
- сформульовано постановку та задачі проекту та основі функції, які повинні бути реалізовані;
- розроблено шаблони для автоматизованих тестів;
- створено фреймворк для автоматизованого тестування та інтегровано із системою управління версій Git;
- впроваджено у систему безперервної інтеграції TeamCity автоматичного процесу тестування.

У результаті виконання дипломного проекту було поглиблено теоретичні та практичні навички зі створення гнучкої інфраструктурної системи автоматичного фреймворку, були проаналізовані можливості та засоби налагодження системи управління версіями Git та системи безперервної інтеграції TeamCity.

Були проаналізовані результати виконання, які показали, що впровадження автоматизації тестування у процес розробки – ефективний варіант для економії часових та фінансових ресурсів, а використання інфраструктурних рівнів автоматизованого фреймворку дозволяє створювати гнучкий та легкопідтримуваний код.

ПЕРЕЛІК ПОСИЛАНЬ

1. Святослав Кулик. Тестирование программного обеспечения. Базовый курс -- 2 издание, Минск, 2015 – 296 с.

2. Standard Glossary of Terms used in Software Testing. *International Software Testing Qualifications Board*. 82 с. URL: https://www.istqb.org/downloads/send/20-istqb-glossary/186-glossary-all-terms.html.

3. ГОСТ 28806–90. Качество программных средств. Термины и определения. [Электронный ресурс] URL: http://docs.cntd.ru/document/5200224, свободный. - Загл. С экрана. – Яз. рус.

4. The Art of Software Testing / Glenford J. Myers, Revised and Updated by Tom Badgett, Todd M.Thomas, Corey Sandler. - 2nd ed. - Hoboken, New Jersey.: John Wiley & Sons, Inc., 2004 - 234 p.

5. Р. Калбертсон, К. Браун, Г. Кобб. Быстрое тестирование. - Вильямс, 2004. – 379 с.

6. Сертифицированный тестировщик. Программа обучения Базового уровня. ISTQB – 94с. 2018

7. Автоматизированное тестирование программного обеспечения - основные понятия. // ПроТестинг.RU. - [Электронный ресурс URL: http://www.protesting.ru/automation/, свободный. - Загл. С экрана. – Яз. Рус.

8. Farrell, C. Harrison N. Under the Hood of .NET Memory Management. Simple Talk Publishing. 2011. 213 с. ISBN 978-1-906434-74-8.

9. Пирамида автоматизации и другие геометрические фигуры // AT.Info [Электронный ресурс URL: http://automated-testing.info/t/piramida-avtomatizacz.., свободный. - Загл. С экрана. – Яз. рус.

10. Test Design Considerations // Selenium HQ. Browser automation. [Электронный ресурс] URL:

http://www.seleniumhq.org/docs/06_test_design_conside.., свободный. - Загл. С экрана. – Яз. рус.

11. Автоматизация тестирования: выбор инструмента. // OpenQuality.ru. Качество программного обеспечения. - [Электронный ресурс] URL: http://blog.openquality.ru/tool-choice/, свободный. - Загл. С экрана. – Яз. рус.

12. Алгоритм оценки времени на тестирование. URL: https://doitsmartly.ru/all-articles/sw-testing/102-test-estimation.html

13. Оценка целесообразности внедрения автоматизированного тестирования. URL: https://naukovedenie.ru/PDF/13tvn113.pdf

14. TeamCity Documentation. TeamCity 10.x and 2017.x Documentation. *Confluence*. URL: https://confluence.jetbrains.com/display/TCD10/TeamCity+Documentation.

15. Зенович Д. Тестирование в Mail.Ru Group. Блог компании Mail.Ru Group. *Хабр*. 2013. URL: https://habr.com/company/mailru/blog/165877.

Додаток А

Тексти програмного коду

Система автоматизації тестування фреймворків Web-порталу

(Найменування програми(документа))

DVD-R

(Вид носія даних)

24 арк., 2930 Кб

(Обсяг програми (документа), арк.) Кб)

Київ – 2019 року

Context

```
namespace Context.Forms
{
    public class FeedForm : Form
    {
        public List<Link> NavigationTab =>
            this.WebElement.FindElements(By.XPath(".//ul[@class='nav nav-
tabs']/li/a")).Select(a => new Link(webElement: a)).ToList();

        [FindBy(XPath = ".//input[@class='btn btn-primary btn-md-2']")]
        public Button PostButton;

        [FindBy(XPath = ".//textarea[@class='form-control']")]
        public TextArea PostTextArea;

        public void NavigateToTab(string tabName)
        {
            NavigationTab.First(element => element.GetText == tabName).Click();
        }
    }
}

namespace Context.Forms
{
    public class LoginForm : Form<User>
    {
        [FindBy(XPath = ".//input[@type='email']")]
        [Name("Email")]
        public TextField EmailField;

        [FindBy(XPath = ".//input[@type='password']")]
        [Name("Password")]
        public TextField PasswordField;

        [FindBy(XPath = ".//input[@type='submit']")]
        [Name("Submit")]
        public Button SubmitButton;

        public List<Label> EmailErrorLabels =>
this.WebElement.FindElements(By.XPath(".//div[contains(@class,'form-group label-
floating')]//span[@class='field-validation-error text-danger']"))
            .Select(element => new Label(webElement: element)).ToList();

        public string IsErrorPresent()
        {
            if (this.WebElement.FindElements(By.XPath(".//div[contains(@class,'form-group
label-floating')]")).Any(element => element.GetAttribute("class").Contains("has-error")))
            {
                if (EmailField.GetText.Contains("@"))
                {
                    return "Введите часть адреса после символа \"@\".";
                }
                else
                {
                    return "Адрес электронной почти должен содержать символ \"@\".";
                }
            }
```

```
            return null;
        }

        public void SetUserData(User user)
        {
            Submit(user);
        }
    }
}
namespace Context.Forms
{
    public class ProfileInfoForm : Form
    {
        [FindBy(XPath = ".//div[@class='ui-block-title']/h6")]
        public Label ProfileInfoFormName;

        [FindBy(XPath = ".//ul[@class='widget w-personal-info item-block]")]
        public PersonalDataSection PersonalData;

        public class PersonalDataSection
        {
            public string GetProfileDataInSection(string section)
            {
                return new Label(By.XPath($"//li[span[@class='title' and contains(text(),
'{section}')]]/span[@class='text']")).GetText;
            }
        }
    }
}

namespace Context.Forms
{
    public class RegisterForm : Form<User>
    {
        public string IsErrorPresent()
        {
            if (this.WebElement.FindElements(By.XPath(".//div[contains(@class,'form-group
label-floating')]")).Any(element => element.GetAttribute("class").Contains("has-error")))
                return "Адрес электронной почти должен содержать символ \"@\".";
            return null;
        }

        public List<Label> EmailErrorLabels =>
this.WebElement.FindElements(By.XPath(".//div[contains(@class,'form-group label-
floating')]//span[@class='field-validation-error text-danger']"))
                .Select(element => new Label(webElement: element)).ToList();

        [FindBy(XPath = ".//button[@data-id='SelectRegisterForm']")]
        [Name("Role")]
        public RoleDropdown SelectRoleDropDown;

        [FindBy(XPath = ".//input[@name='FirstName']")]
        [Name("FirstName")]
        public TextField NameField;

        [FindBy(XPath = ".//input[@name='LastName']")]
        [Name("LastName")]
        public TextField LastNameField;
```

```
        [FindBy(XPath = ".//input[@name='Email']")]
        [Name("Email")]
        public TextField EmailField;

        [FindBy(XPath = ".//input[@name='Password']")]
        [Name("Password")]
        public TextField PasswordField;

        [FindBy(XPath = ".//input[@name='ConfirmPassword']")]
        [Name("Password")]
        public TextField ConfirmPasswordField;

        [FindBy(XPath = ".//input[@name='Institution']")]
        [Name("Institution")]
        public TextField InstitutionField;

        [FindBy(XPath = ".//input[@name='Position']")]
        [Name("Position")]
        public TextField PositionField;

        [FindBy(XPath = ".//input[@type='submit']")]
        [Name("Submit")]
        public Button SubmitButton;

        public void SetUserData(User user)
        {
            SelectRoleDropDown.SelectRole(user.Role);
            NameField.SendKeys(user.FirstName);
            LastNameField.SendKeys(user.LastName);
            EmailField.SendKeys(user.Email);
            PasswordField.SendKeys(user.Password);
            ConfirmPasswordField.SendKeys(user.Password);
            InstitutionField.SendKeys(user.Institution);
            if(user.Role != UserRole.Student)
                PositionField.SendKeys(user.Position);
        }

        public void SubmitUser(User user)
        {
            SetUserData(user);
            SubmitButton.Click();
        }
    }
}

namespace Context.Forms
{
    public class RegistrationLoginForm : Form
    {
        [FindBy(XPath = ".//form[@action='/Account/Login']")]
        public LoginForm LoginForm;

        [FindBy(XPath = ".//form[@action='/Account/Register']")]
        public RegisterForm RegisterForm;

        [FindBy(XPath = ".//a[contains(text(),'Реєстрація')]")]
        [Name("Register")]
        public Button RegisterTab;
```

```
        [FindBy(XPath = ".//a[contains(text(),'Вхід')]")]
        [Name("Login")]
        public Button LoginTab;

        public void SelectForm(string form)
        {
            var button = GetElementClass.GetButton(form);
            if (!button.GetAttribute("class").Contains("active"))
                button.Click();
        }
    }
}


namespace Context.Pages
{
    public class AboutPage : WebPage
    {
        [FindBy(XPath = ".//div[@class='stunning-header-content']")]
        public AboutPageNavigationSection AboutPageNavigationSection;

        [FindBy(XPath = ".//div[@class='header--standard-wrap']")]
        public AboutPageHeaderSection AboutPageHeaderSection;
    }
}

namespace Context.Pages
{
    public class ContactPage : WebPage
    {
        [FindBy(XPath = ".//a[@class='btn btn-md btn-border c-white']")]
        public Button GoToLoginPageButton;

        [FindBy(XPath = ".//ul[@role='tablist']")]
        public ContactSection ContactSection;
    }
}

namespace Context.Pages
{
    public class DefaultPage : WebPage
    {
        [FindBy(XPath = ".//a[contains(text(),'Стрічка новин')]")]
        public Button ProfileButton;

        [FindBy(XPath = ".//a[contains(text(),'Про проект')]")]
        public Button ProjectButton;

        [FindBy(XPath = ".//a[contains(text(),'Спільнота')]")]
        public LabelWithList CommunityButton;

        [FindBy(XPath = ".//a[contains(text(),'Галерея')]")]
        public Button GalleryButton;

        [FindBy(XPath = ".//a[contains(text(),'Події')]")]
        public Button EventsButton;

        [FindBy(XPath = ".//span[contains(text(),'Інформація користувача')]")]
        public LabelWithList UserInfoLabel;
```

```
        [FindBy(ClassName = "modal-dialog window-popup edit-widget edit-widget-blog-post")]
        public Popup NotImplementedPopup;

        [FindBy(XPath = ".//header[@id='site-header']")]
        public HeaderSection HeaderSection;
    }
}
namespace Context.Pages
{
    public class LoginPage : WebPage
    {
        [FindBy(XPath = ".//div[@class='registration-login-form']")]
        public RegistrationLoginForm RegistrationLoginForm;

        [FindBy(XPath = "//a[contains(text(),'Про проект')]")]
        public Button AboutProjectLink;

        [FindBy(XPath = ".//div[@class='header--standard-wrap']")]
        public LoginHeaderSection HeaderSection;

        [FindBy(XPath = ".//input[@type='submit']")]
        [Name("Submit")]
        public Button SubmitButton;

        public Button GetControlPanelButton(string buttonName)
        {
            return HeaderSection.ControlButton.First(element => element.GetText ==
buttonName);
        }

        public string GetErrorMessageForRegistration(string type)
        {
            if(RegistrationLoginForm.RegisterForm.IsErrorPresent() != null)
            {
                return RegistrationLoginForm.RegisterForm.IsErrorPresent();
            }
            else
            {
                var elements = RegistrationLoginForm.RegisterForm.EmailErrorLabels;
                if(!elements.Any())
                    throw new Exception("Unable to find any error message!");
                var label = elements.First(element => element.GetAttribute("data-valmsg-
for") == type & element.Displayed);
                    return label.GetText;
            }
        }

        public string GetErrorMessageForLogin(string type)
        {
            if (RegistrationLoginForm.LoginForm.IsErrorPresent() != null)
            {
                return RegistrationLoginForm.LoginForm.IsErrorPresent();
            }
            else
            {
                var elements = RegistrationLoginForm.LoginForm.EmailErrorLabels;

                if (!elements.Any())
```

```
                    throw new Exception("Unable to find any error message!");

                var label = elements.First(element => element.GetAttribute("data-valmsg-
for") == type);
                    return label.GetText;
            }
        }

        public void ClickOnExpandCollapseButton()
        {
            HeaderSection.ExpandCollapseButton.Click();
        }

        public bool IsLoginMenuDisplayed()
        {
            if (HeaderSection.ControlButton.First().Displayed)
                return true;
            return false;
        }

        public void ClickOnGoToLoginPage()
        {
            WebPortal.ContactPage.GoToLoginPageButton.Click();
        }

        public void SetTextToRegisterField(string fieldName, string text)
        {
           var field =
this.RegistrationLoginForm.RegisterForm.WebElement.FindElement(By.Name(fieldName));
            field.SendKeys(text);
        }
    }
}

namespace Context.Pages
{
    public class NewsPage : DefaultPage
    {
        [FindBy(XPath = ".//div[@class='news-feed-form']")]
        public FeedForm FeedForm;

        [FindBy(XPath = ".//div[@class='ui-block']")]
        public ProfileInfoForm ProfileInfoForm;

        [FindBy(XPath = ".//div[@id='newsfeed']")]
        public AllPostSection AllPostSection;

        public void SelectSettingItem(string buttonName)
        {
            var label = HeaderSection.UserLabel;
            label.MouseOver();
            this.HeaderSection.UserMenuSection.SettingsSection.SelectButton(buttonName);
        }
    }
}namespace Context.Sections
{
    public class AboutPageHeaderSection : Section
    {
```

```
        public List<Button> AboutPageHeaderButtons =>
this.WebElement.FindElements(By.XPath(".//li[@class='nav-item']/a"))
                .Select(element => new Button(webElement: element)).ToList();

        public Button CollapseExpandMenu => new Button(webElement:
WebElement.FindElement(By.XPath("//li[@class='nav-item js-expanded-menu']")));

        [FindBy(XPath = ".//li[@class='nav-item dropdown']")]
        public AboutPageDropdown AboutPageDropdown;

        public Label AboutProfileLabel => new Label(webElement:
            WebElement.FindElement(By.XPath("//a[@class='nav-link dropdown-toggle' and
text()='Профіль' and @data-hover='dropdown']")));

        public List<Button> Buttons =>
            WebElement.FindElements(By.XPath(".//div[@class='dropdown-menu']/a"))
            .Select(element => new Button(webElement: element)).ToList();

        public void ClickOnProfileMenuItem(string buttonName)
        {
            if (!AboutPageDropdown.Displayed)
                this.CollapseExpandMenu.Click();

            //var label = this.AboutProfileLabel;
            //label.MouseOver();

            //var buttons = this.Buttons;
            //buttons.First(button => button.GetText == buttonName).Click();
            AboutPageDropdown.SelectItem(buttonName);
        }

    }
}

namespace Context.Sections
{
    public class AdditionalPostDataSection : Section
    {
        [FindBy(XPath = ".//a[@class='post-add-icon inline-items add-like active']")]
        public Button Like;

        public List<Label> UsersWhoLikesLabels =>
            WebElement.FindElements(By.XPath(".//div[@class='names-people-likes']/a"))
            .Select(element => new Label(webElement: element)).ToList();

        [FindBy(XPath = ".//a[@class='post-add-icon inline-items comment-href ']/span")]
        [Name("Comment")]
        public Label AmoutOfCommentsLabel;

        [FindBy(XPath = ".//a[@class='post-add-icon inline-items repost-href']/span")]
        [Name("Repost")]
        public Button AmountOfRepostLabel;

        public bool IsPostLiked()
        {
            var postState = Like.GetAttribute("class").Split(' ').Last();

            if (postState == "active")
                return true;
```

```
            return false;
        }

        public string GetAmountOfLikes()
        {
            var amount = (Label)Like.GetWebElement().FindElement(By.XPath(".//span"));

            return amount.GetText;
        }

        public void GetUserNamesWhoLikes()
        {
            var userNames = UsersWhoLikesLabels.Select(user => user.GetText);

        }

        public string GetAmountOf(string actions)
        {
            var label = GetElementClass.GetButton(actions);
            return label.GetText;
        }

    }
}

namespace Context
{
   public static class ITPortalContext
    {
        public static string IsNewsPageOpen()
        {
            var newsPage = WebPortal.DefaultPage;
            return GetPageHeader();
        }

        public static bool IsLoginPageOpened()
        {
            var loginPage = WebPortal.LoginPage;
            var logo = loginPage.HeaderSection.LoginLogo;
            if (logo.Displayed)
                return true;
            return false;
        }

        public static bool IsAboutPageOpened()
        {
            var aboutPageTitle = WebPortal.AboutPage.AboutPageNavigationSection.PageTitle;
            if (aboutPageTitle.Displayed)
                return true;
            return false;
        }

        public static string GetPageHeader()
        {
            return WebPortal.DefaultPage.HeaderSection.PageTitleLabel.GetText;
        }

        public static void SelectTabInUserInfo(string button)
        {
```

```
            WebPortal.DefaultPage.UserInfoLabel.MouseOver();
            WebPortal.DefaultPage.UserInfoLabel.SelectElementFromDropdown(button);
        }

    }
}

namespace Context
{
    [Site(Domain = "http://pidhulko.1gb.ua")]
    public class WebPortal : JDI_Web.Selenium.Elements.Composite.WebSite
    {
        [Page(Title = "IT knowledge")]
        public static LoginPage LoginPage;

        [Page(Url = "/Profile", Title = "Стрічка новин")]
        public static NewsPage NewsPage;

        [Page(Title = "Моя сторінка")]
        public static ProfilePage ProfilePage;

        [Page(Url = "/Home/Contact")]
        public static ContactPage ContactPage;

        //[Page(Url = "/CommunityList", Title = "Спільнота")]
        //public static CommunityPage CommunityPage;

        [Page(Url = "/Profile/Edit", Title = "Редагування профайлу")]
        public static EditProfilePage EditProfilePage;

        [Page(Url = "/AboutProject", Title = "Про проект")]
        public static AboutPage AboutPage;

        public static DefaultPage DefaultPage;

    }
}
```

Core

```
namespace Core.DTO
{
    public class ExtendedUser : User
    {
        public (string,string) Object { get; set; }
    }

    public class ListExtendedUsersWithMessage : ITestDataSource
    {
        [XmlElement(ElementName = "User")]
        public ExtendedUser[] Users { get; set; }
    }
 }

namespace Core.DTO
{
    public interface ITestDataSource
    {
```

```
    }
}

namespace Core.DTO
{
    public class ListUsers : ITestDataSource
    {
        [XmlElement(ElementName = "User")]
        public User[] Users { get; set; }
    }
}

namespace Core.DTO
{
    public class ListUsersWithMessage : ITestDataSource
    {
        [XmlElement(ElementName = "User")]
        public User[] Users { get; set; }

        [XmlElement(ElementName ="Message")]
        public (string, string)[] Messages { get; set; }
    }
}
namespace Core.Elements
{
    public class RoleDropdown : Dropdown
    {
        public Button ClickableElement => new Button(webElement:
WebElement.FindElement(By.XPath(".//span[@class='filter-option pull-left']")));

        public List<Button> Roles =>
            this.WebElement.FindElements(By.XPath("//ul[@class='dropdown-menu
inner']/li/a/span[@class='text']"))
            .Select(element => new Button(webElement: element)).ToList();

        public void SelectRole(UserRole? role)
        {
            string roleToString= null;
            switch (role)
            {
                case UserRole.Company:
                    roleToString = "Представник компанії";
                    break;
                case UserRole.Univercity:
                    roleToString = "Співробітник університету";
                    break;
                case UserRole.Student:
                    roleToString = "Студент";
                    break;
            }

            this.ClickableElement.Click();
            this.Roles.First(element => element.GetText == roleToString).Click();
        }
    }

    public class AboutPageDropdown : Dropdown
    {
```

```
        public Label AboutProfileLabel => new Label(webElement:
WebElement.FindElement(By.XPath("//a[@class='nav-link dropdown-toggle' and text()='Профіль'
and @data-hover='dropdown']")));
        public List<Button> AboutProfileLabels =>
    this.WebElement.FindElements(By.XPath("//a[@class='nav-link dropdown-toggle' and
text()='Профіль' and @data-hover='dropdown']"))
    .Select(element => new Button(webElement: element)).ToList();

        public List<Button> Buttons =>
            WebElement.FindElements(By.XPath(".//div[@class='dropdown-menu']/a"))
            .Select(element => new Button(webElement: element)).ToList();

        public void SelectItem(string buttonName)
        {

            AboutProfileLabel.MouseOver();

            Buttons.First(button => button.GetText == buttonName).Click();
        }
    }
}
namespace Core.Elements
{
    public class LabelWithList : Label
    {
        public void SelectElementFromDropdown(string link)
        {
            new Button(By.XPath($".//ul[@class='more-dropdown more-with-
triangle']/li/a[contains(text(), '{link}')]")).Click();
        }
    }
}
namespace Core.Elements
{
    public class Popup: Text, IPopup, IText, IHasValue, IElement, IBaseElement, IVisible,
IComposite
    {

        public void Ok()
        {
            new Link(By.XPath(".//a[@class='btn btn-primary btn-lg full-width']")).Click();
        }

        public void Cancel()
        {
            new Link(By.XPath(".//a[@class='close icon-close' and @aria-
label='Close']")).Click();
        }

        public void Close()
        {
            Cancel();
        }

        public string GetPopupText => new Label(By.XPath("//h3[@id='modal-info-result-
text']")).GetText;

    }
}
```

```
namespace Core.Extensions
{
    public static class XMLExtensions
    {
        public static T FromXML<T>(this T obj, string fileName) where T : ITestDataSource
        {
            var serializer = new XmlSerializer(typeof(T));
            var fullFilePath = ComposeFilePath(fileName);
            var reader = new StreamReader(fullFilePath);
            var result = (T)serializer.Deserialize(reader);
            return result;
        }

        public static void ToXML<T>(this T obj, string fileName) where T : ITestDataSource
        {
            var fullFilePath = ComposeFilePath(fileName);
            new XmlDocument().CreateElement(fileName);
            var serializer = new XmlSerializer(typeof(T));

            var txtWriter = new StreamWriter(fullFilePath);

            serializer.Serialize(txtWriter, obj);

            txtWriter.Close();
        }

        private static string ComposeFilePath(string fileName)
        {
            var buildDirectoryPath = Path.GetDirectoryName(
                Assembly.GetExecutingAssembly().Location
                );
            var fullFilePath = Path.Combine(buildDirectoryPath,"CommonData", fileName);
            return fullFilePath;
        }
    }
}
namespace Core
{
    public class SessionService
    {
        private static readonly Dictionary<string, TestrunContext> ContextsStorage = new
Dictionary<string, TestrunContext>();

        public static TestrunContext Current
        {
            get
            {
                TestrunContext value;
                var result = ContextsStorage.TryGetValue(TestContext.CurrentContext.Test.ID,
out value);

                if (result == true)
                    return value;
                throw new Exception("Context for current session is missing in context
storage. Have you forgotten to start one?");
            }
        }

        public static void StartSession()
```

```
        {
            var parentId = GetParentContextId();

            try
            {
                ContextsStorage.Add(TestContext.CurrentContext.Test.ID, new
TestrunContext(parentId));
            }
            catch (ArgumentException ex)
            {
                throw new ArgumentException("Session with given ID was already in storage.
Have you tried to start it few times?", ex);
            }
        }

        public static void CloseSession()
        {
            foreach (var action in Current.TearDown)
                action.Invoke();

            ContextsStorage.Remove(TestContext.CurrentContext.Test.ID);
        }

        private static string GetParentContextId()
        {
            var testFieldInfo = TestContext.CurrentContext.Test.GetType()
                .GetField("_test", BindingFlags.NonPublic | BindingFlags.Instance);

            if (testFieldInfo != null
                && testFieldInfo.GetValue(TestContext.CurrentContext.Test) is
NUnit.Framework.Internal.Test)
                return
((Test)testFieldInfo.GetValue(TestContext.CurrentContext.Test)).Parent.Id;

            throw new NUnitException("Failed reflecting to get parent`s context Id.");
        }
    }
}
namespace Core
{
    public class TestrunContext
    {
        public string ParentContextId { get; set; }

        public User CurrentUser { get; set; }

        internal Stack<Action> TearDown { get; } = new Stack<Action>();

        public TestrunContext(string parentContextId)
        {
            ParentContextId = parentContextId;
        }
    }
}namespace Core
{
    public class User
    {
        [Name("User role")]
        public UserRole? Role { get; set; }
```

```
        [Name("First name")]
        public string FirstName { get; set; }

        [Name("Last name")]
        public string LastName { get; set; }

        [Name("Password")]
        public string Password { get; set; }

        [Name("Email")]
        public string Email { get; set; }

        [Name("Position")]
        public string Position { get; set; }

        [Name("Institution")]
        public string Institution { get; set; }
    }
}
namespace Core
{
    public enum UserRole
    {
        Univercity,
        Company,
        Student
    }
}
```

Steps

```
namespace Steps
{
    public class CommonSteps
    {
        public void LoginToPortalAs(User user)
        {
            var loginPage = WebPortal.LoginPage.RegistrationLoginForm;
            loginPage.SelectForm("Login");
            loginPage.LoginForm.SetUserData(user);
            SessionService.Current.CurrentUser = user;
        }

        public void LoginToPortalAsDefaultUser()
        {

            var user = new User() { Email = "poncik@gmail.com", Password = "Password1" };
            LoginToPortalAs(user);
        }

        public void SetUserDataOnRegisterForm(User user)
        {
            var registerPage = WebPortal.LoginPage.RegistrationLoginForm;
            registerPage.SelectForm("Register");
            registerPage.RegisterForm.SetUserData(user);
        }
```

```
public void SubmitUserDataOnRegisterForm(User user)
{
    var registerPage = WebPortal.LoginPage.RegistrationLoginForm;
    registerPage.RegisterForm.SubmitUser(user);

}

public void GoToFromControlPanel(string buttonName)
{
    var button = WebPortal.LoginPage.GetControlPanelButton(buttonName);
    button.Click();
}

public string GetErrorMessageForFielsInForm(string field, string form)
{
    if(form == "Login")
    {
        return WebPortal.LoginPage.GetErrorMessageForLogin(field);
    }
    else
    {
        return WebPortal.LoginPage.GetErrorMessageForRegistration(field);
    }
}

public void CollapseLoginMenu()
{
    WebPortal.LoginPage.ClickOnExpandCollapseButton();
}

public bool IsLoginMenuCollapsed()
{
    if (WebPortal.LoginPage.IsLoginMenuDisplayed())
    {
        return false;
    }
    return true;
}

public void GoToLoginPageFromContactPage()
{
    WebPortal.LoginPage.ClickOnGoToLoginPage();
}

public void GoToPageFromAboutPage(string buttonName)
{
    var dropdown = WebPortal.AboutPage.AboutPageHeaderSection;
    dropdown.ClickOnProfileMenuItem(buttonName);
}

public void SetTextToFieldOnRegisterForm(string field, string text)
{
    WebPortal.LoginPage.SetTextToRegisterField(field, text);

}

public void SubmitChanges()
{
    WebPortal.LoginPage.SubmitButton.Click();
```

```
        }
    }
}
namespace Steps
{
    public class NewsAndProfileSteps
    {


        public void LogoutFromPortal()
        {
            WebPortal.NewsPage.SelectSettingItem("Вихід");
        }
    }
}
```

Test

```
namespace Test.CommonData
{
    public static class TestExtendedData
    {
        //public static IEnumerable<User> TestDataUser = new
ListUsersWithMessage().FromXML("TestListUsersWithMessage.xml").Users;


        public static IEnumerable<ExtendedUser> TestDataExtendedUserLogin = new
ListExtendedUsersWithMessage().FromXML("TestListExtendedUsersWithMessageForLogin.xml").Users
;
        public static IEnumerable TestCasesLogin
        {
            get
            {
                foreach (var testData in TestDataExtendedUserLogin)
                {
                    var data = new TestCaseData(testData);
                    data.TestName =
$"LoginForm_EnterWrongCreadentials_{testData.Email}_{testData.Password}";
                    yield return data;
                }
            }
        }

        public static IEnumerable<ExtendedUser> TestDataExtendedUserRegister = new
ListExtendedUsersWithMessage().FromXML("TestListExtendedUsersWithMessageForRegister.xml").Us
ers;
        public static IEnumerable TestCasesRegister
        {
            get
            {
                foreach (var testData in TestDataExtendedUserRegister)
                {
                    var data = new TestCaseData(testData);
                    data.TestName =
$"RegisterForm_EnterWrongCredentials_{testData.FirstName}_{testData.Password}";
                    yield return data;
                }
            }
        }
```

```
    }
}

namespace Test
{
    public class BaseTestFixture
    {
        [OneTimeSetUp]
        [Order(0)]
        public void FeatureSetup()
        {
            SessionService.StartSession();
        }

        [SetUp]
        [Order(0)]
        public void TestSetup()
        {
            SessionService.StartSession();
        }

        [TearDown]
        [Order(0)]
        public void TestTeardown()
        {
            SessionService.CloseSession();
        }

        [OneTimeTearDown]
        [Order(0)]
        public void FeatureTeardown()
        {
            SessionService.CloseSession();
        }
    }
}

namespace Test
{
    [SetUpFixture]
    class Hooks
    {
        [OneTimeSetUp]
        public static void GlobalSetup()
        {
            log4net.Config.XmlConfigurator.Configure();
            WebSettings.InitNUnitDefault();
            var time = new TimeoutSettings();
            time.WaitElementSec = 30;
            WebSettings.Timeouts = time;
            WinProcUtils.KillAllRunWebDrivers();
            WebSite.Init(typeof(WebPortal));
            WebPortal.Open();
        }

        [OneTimeTearDown]
        public void GlobalTeardown()
        {
            WinProcUtils.KillAllRunWebDrivers();
        }
```

```
    }
}

namespace Test
{
    [TestFixture]
    [Category("Login Page")]
    public class LoginPageFunctionality : BaseTestFixture
    {
        public static CommonSteps _loginSteps = new CommonSteps();


        //public static IEnumerable<ExtendedUser> TestDataExtendedUser = new
ListExtendedUsersWithMessage().FromXML("TestListExtendedUsersWithMessage.xml").Users;



        [Test]
        [TestCaseSource(typeof(TestExtendedData), "TestCasesLogin")]
        public void LoginForm_EnterWrongCredentials(ExtendedUser user)
        {
            _loginSteps.LoginToPortalAs(user);
            var actualMessage = _loginSteps.GetErrorMessageForFielsInForm(user.Object.Item1,
"Login");
            var expectedMessage = user.Object.Item2;

            Assert.That(actualMessage, Is.EqualTo(expectedMessage), "Actual message does not
equal to expected!");
        }

        [Test]
        [TestCaseSource(typeof(TestExtendedData), "TestCasesRegister")]
        public void RegisterForm_EnterWrongCredentials(ExtendedUser user)
        {
            _loginSteps.SubmitUserDataOnRegisterForm(user);
            var actualMessage = _loginSteps.GetErrorMessageForFielsInForm(user.Object.Item1,
"Register");
            var expectedMessage = user.Object.Item2;

            Assert.That(actualMessage, Is.EqualTo(expectedMessage), "Actual message does not
equal to expected!");
        }

        [Test]
        public void LoginPage_CollapseMenu()
        {
            _loginSteps.CollapseLoginMenu();
            var actualState = _loginSteps.IsLoginMenuCollapsed();

            Assert.That(actualState, Is.True, "Login Menu is not collapsed!");
        }

        [Test]
        public void LoginPage_NavigateToLoginPageFromContact()
        {
            _loginSteps.GoToFromControlPanel("Контакти");
            _loginSteps.GoToLoginPageFromContactPage();

            var actualIsOpened = ITPortalContext.IsLoginPageOpened();
```

```
            Assert.That(actualIsOpened, "The Login Page isn't opened!");
        }

        [Test]
        public void LoginPage_NavigateToAboutPageFromLoginPage()
        {
            _loginSteps.GoToFromControlPanel("Про проект");

            var actualIsOpened = ITPortalContext.IsAboutPageOpened();

            Assert.That(actualIsOpened, "The Login Page isn't opened!");
        }

        [Test]
        public void AboutPage_UnauthorizedEntry()
        {
            _loginSteps.GoToFromControlPanel("Про проект");
            _loginSteps.GoToPageFromAboutPage("Мій профіль");

            var actualIsOpened = ITPortalContext.IsLoginPageOpened();

            Assert.That(actualIsOpened, "The Login Page isn't opened!");
        }

        [Test]
        public void LoginPage_EnterMismatchPassword_ErrorMessage()
        {
            _loginSteps.SetTextToFieldOnRegisterForm("Password", "password");
            _loginSteps.SetTextToFieldOnRegisterForm("ConfirmPassword", "password1");
            _loginSteps.SubmitChanges();

            var actualMessage = _loginSteps.GetErrorMessageForFielsInForm("ConfirmPassword",
"Register");
            var expectedMessage = "Паролі не співпадають.";

            Assert.That(actualMessage, Is.EqualTo(expectedMessage), "Confirm Password field
does not have expected error message!");

        }
    }
}
namespace Test
{
    [TestFixture]
    [Category("Smoke")]
    public class Smoke : BaseTestFixture
    {

        //public static IEnumerable<User> TestData = new
ListUsers().FromXML("TestUsersList.xml").Users;
        public static CommonSteps _loginSteps = new CommonSteps();
        public static NewsAndProfileSteps _newsAndProfileSteps = new NewsAndProfileSteps();

        [Test]
        public void ITPortal_Login()
        {
            var expectedHeader = "Стрічка новин";
```

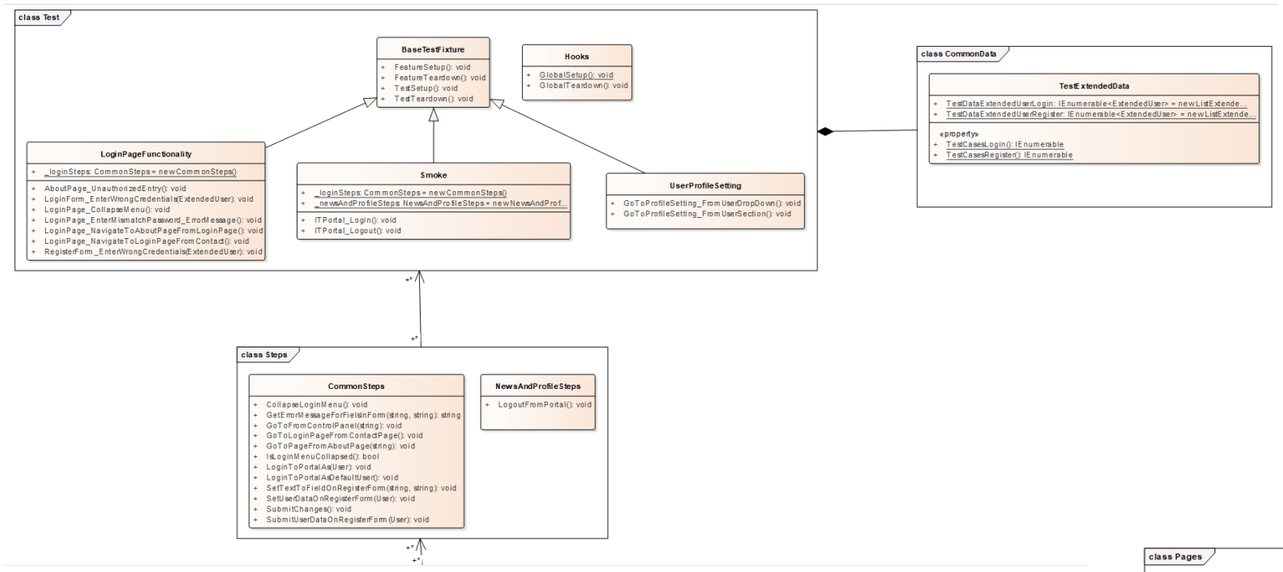
```
            _loginSteps.LoginToPortalAsDefaultUser();
            var actualHeader = ITPortalContext.IsNewsPageOpen();

            Assert.That(actualHeader, Is.EqualTo(expectedHeader.ToUpperInvariant()), "The
News Page isn't opened!");
        }

        [Test]
        public void ITPortal_Logout()
        {
            _loginSteps.LoginToPortalAsDefaultUser();
            _newsAndProfileSteps.LogoutFromPortal();

            var actualIsOpened = ITPortalContext.IsLoginPageOpened();

            Assert.That(actualIsOpened, "The Login Page isn't opened!");
        }
    }
}
namespace Test
{
    [TestFixture]
    [Category("User setting")]
    public class UserProfileSetting : BaseTestFixture
    {
        [Test]
        public void GoToProfileSetting_FromUserDropDown()
        {
     _loginSteps.GoToFromControlPanel("Про проект");
            _loginSteps.GoToPageFromAboutPage("Мій профіль");

            var actualIsOpened = ITPortalContext.IsLoginPageOpened();

            Assert.That(actualIsOpened, "The Login Page isn't opened!");

        }

        [Test]
        public void GoToProfileSetting_FromUserSection()
        {
     _loginSteps.SetTextToFieldOnRegisterForm("Password", "password");
            _loginSteps.SetTextToFieldOnRegisterForm("ConfirmPassword", "password1");
            _loginSteps.SubmitChanges();

            var actualMessage = _loginSteps.GetErrorMessageForFielsInForm("ConfirmPassword",
"Register");
            var expectedMessage = "Паролі не співпадають.";

            Assert.That(actualMessage, Is.EqualTo(expectedMessage), "Confirm Password field
does not have expected error message!");


        }

    }
}
namespace Test
{
    public class XMLSourceExample : BaseTestFixture
```

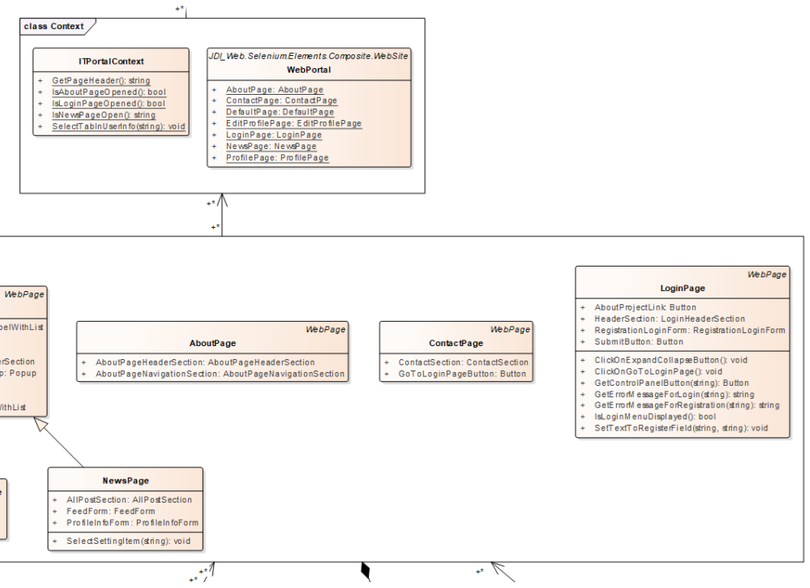
```
    {
        public List<(string, string)> list = new List<(string, string)> { ("", ""), ("", "")
};
        public ListUsers UsersExample = new ListUsers()
        {
            Users = new User[]
            {
                new User()
                {
                    Email = "bla",
                    FirstName = "bla",
                    Institution = "bla",
                    LastName = "bla",
                    Password = "bla",
                    Position = "bla",
                    Role = UserRole.Student
                },
                new User()
                {
                    Email = "bla",
                    FirstName = "bla",
                    Institution = "bla",
                    LastName = "bla",
                    Password = "bla",
                    Position = "bla",
                    Role = UserRole.Student
                },
                new User()
                {
                    Email = "bla",
                    FirstName = "bla",
                    Institution = "bla",
                    LastName = "bla",
                    Password = "bla",
                    Position = "bla",
                    Role = UserRole.Student
                }
            }
        };
        public ListUsersWithMessage usersmess = new ListUsersWithMessage()
        {
            Users = new User[]
            {
                new User()
                {
                    Email = "bla",
                    FirstName = "bla",
                    Institution = "bla",
                    LastName = "bla",
                    Password = "bla",
                    Position = "bla",
                    Role = UserRole.Student
    },
                new User()
                {
                    Email = "bla",
                    FirstName = "bla",
                    Institution = "bla",
                    LastName = "bla",
```

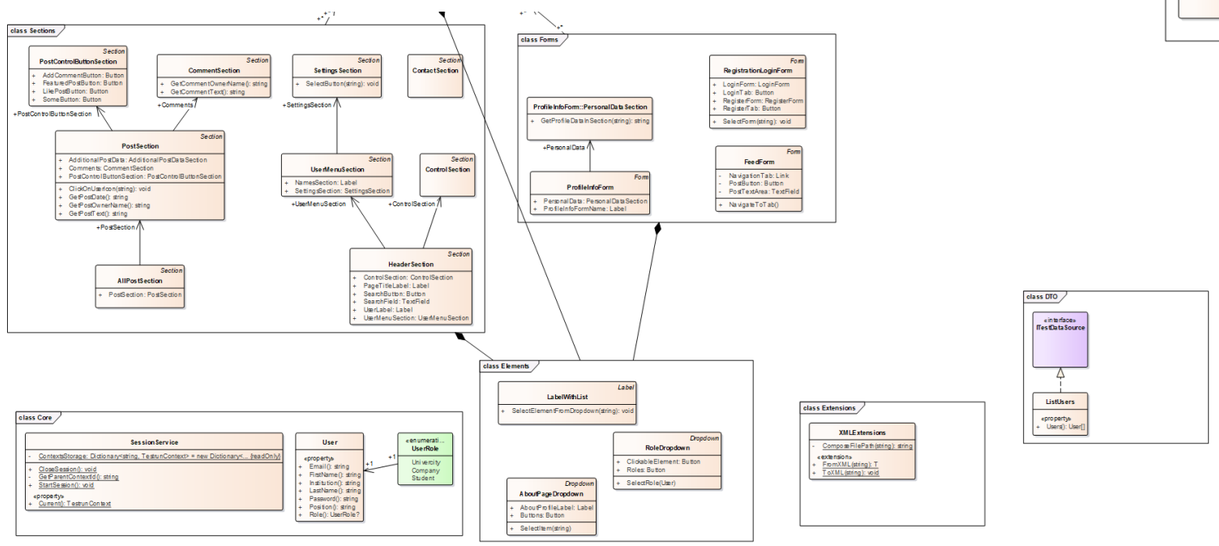
```
                    Password = "bla",
                    Position = "bla",
                    Role = UserRole.Student
                },
                new User()
                {
                    Email = "bla",
                    FirstName = "bla",
                    Institution = "bla",
                    LastName = "bla",
                    Password = "bla",
                    Position = "bla",
                    Role = UserRole.Student
                }
            },
            Messages = new (string, string)[]
              {
                  ("Email", "cow"),
                  ("Password", "chickens"),
                  ("Email", "airplane")
              }
        };

        public ListExtendedUsersWithMessage UsersExample1 = new
ListExtendedUsersWithMessage()
        {
            Users = new ExtendedUser[]
    {
                new ExtendedUser()
                {
                    Email = "",
                    FirstName = "Name",
                    Institution = "institu",
                    LastName = "Last Name",
                    Password = "password",
                    Position = "position",
                    Role = UserRole.Student,
                    Object =  ("Email","Це поле є обов'язкове." )

                },
            new ExtendedUser()
                {
                    Email = "test@test.com",
                    FirstName = "",
                    Institution = "institu",
                    LastName = "Last Name",
                    Password = "password",
                    Position = "position",
                    Role = UserRole.Student,
                    Object =  ("FirstName","Це поле є обов'язкове." )

                },
            new ExtendedUser()
                {
                    Email = "test@test.com",
                    FirstName = "Name",
                    Institution = "",
                    LastName = "Last Name",
                    Password = "password",
```

```
                    Position = "position",
                    Role = UserRole.Student,
                    Object =  ("Institution","Це поле є обов'язкове." )

                },
            new ExtendedUser()
                {
                    Email = "test@test.com",
                    FirstName = "Name",
                    Institution = "institu",
                    LastName = "",
                    Password = "password",
                    Position = "position",
                    Role = UserRole.Student,
                    Object =  ("LastName","Це поле є обов'язкове." )

                },
            new ExtendedUser()
                {
                    Email = "test@test.com",
                    FirstName = "Name",
                    Institution = "institu",
                    LastName = "Last Name",
                    Password = "",
                    Position = "position",
                    Role = UserRole.Student,
                    Object =  ("Password","Це поле є обов'язкове." )

                },

    }
        };

        [Test]
        [Category("Ignore")]
        public void PrepareXML()
        {
            UsersExample1.ToXML("TestListExtendedUsersWithMessageForRegister.xml");
        }

        [Test]
        [TestCaseSource("TestData")]
        [Category("Ignore")]
        public void TestAttribute(User user)
        {
            SessionService.Current.CurrentUser = user;

            WebPortal.LoginPage.Open();
            var login = WebPortal.LoginPage.RegistrationLoginForm.LoginForm;
            WebPortal.LoginPage.RegistrationLoginForm.SelectForm("Login");
            login.Submit(user);
        }
    }
}
```

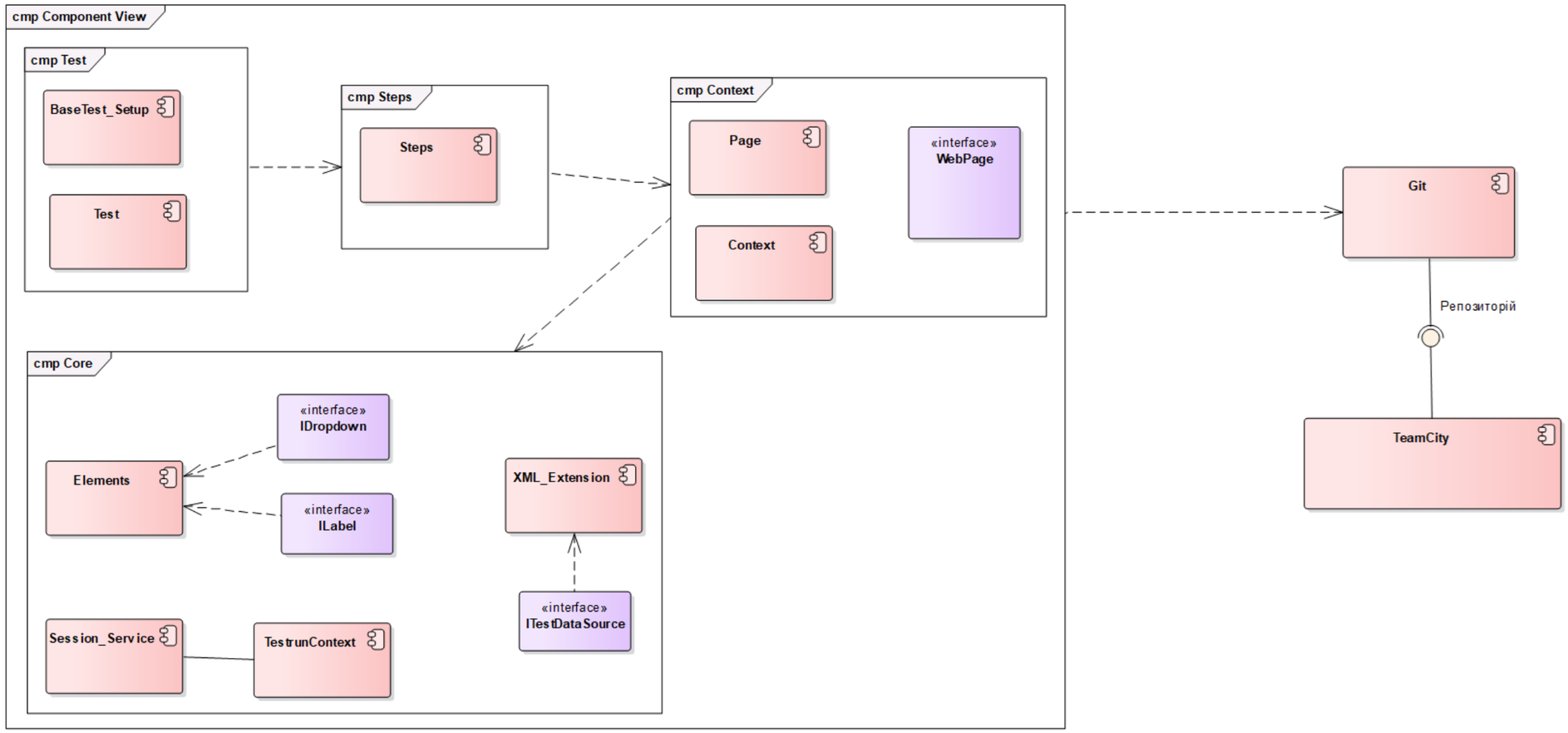
ДП ІС-5115.1260-с.ССК



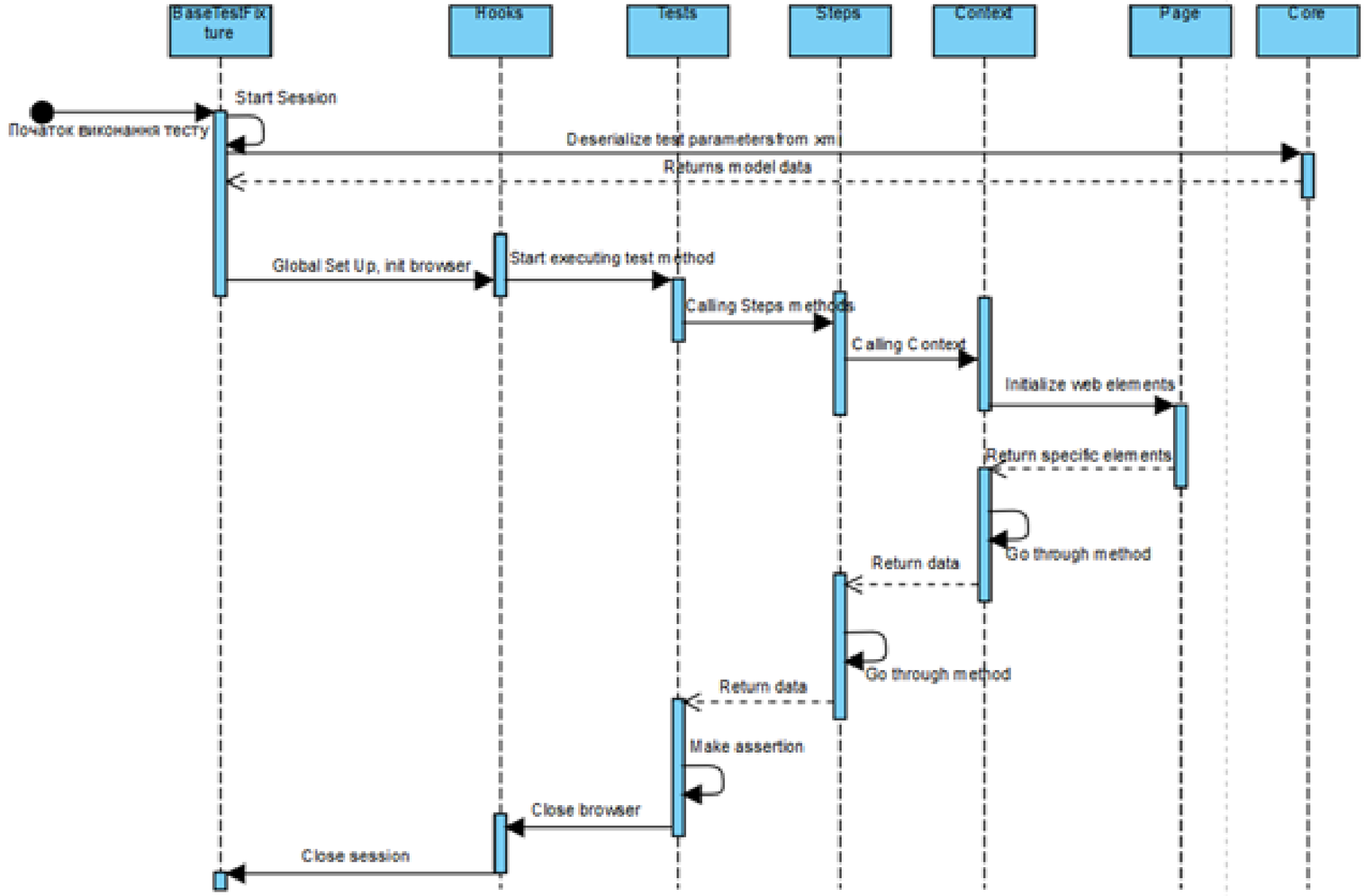




					ДП ІС-5115.1260-с.ССК			
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів	Літера	Маса	Масштаб
Розробив		Устенко Ю.Т.						
Перевірив		Новікова П.А.				Аркуш 1	Аркушів 1	
Т. кон.								
					Система автоматизації тестування фреймворків Web-порталу	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-з51		
Н. кон.		Тєлишева Т.О.						
Затвердив		Новікова П.А.						



					ДП ІС-5115.1260-с.ССК		
					Схема структурна компонентів	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Устенко Ю.Т.				Аркуш 1	Аркушів 1
Перевірив		Новікова П.А.					
Т. кон.							
Н. кон.		Тєлишева Т.О.			Система автоматизації тестування фреймворків Web-порталу	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-з51	
Затвердив		Новікова П.А.					



					ДП ІС-5115.1260-с.ССП		
Зм.	Арк.	№ документа	Підпис	Дата	Схема послідовності роботи розробленої системи	Літера	Маса / Масштаб
Розробив		Устенко Ю.Т.					
Перевірив		Новікова П.А.				Аркуш 1	Аркушів 1
Т. кон.							
Н. кон.		Тєлишева Т.О.			Система автоматизації тестування фреймворків Web-порталу	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-351	
Затвердив		Новікова П.А.					

пн 03.06.2019 00:13

itportaltest2019@gmail.com

[TeamCity, FAILED] Build DiplomAutomation / Test Run [refs/heads/master] #52

Кому Yuliia Ustenko

Build **DiplomAutomation / Test Run** [refs/heads/master] #52 failed (Tests failed: 5, passed: 4, ignored: 11)
Agent: WindowsAgent

Failed tests summary: 5

Test.dll: Test.LoginPageFunctionality.LoginPage_CollapseMenu	details »
Test.dll: Test.LoginPageFunctionality.LoginPage_EnterMismatchPassword_ErrorMessage	details »
Test.dll: Test.LoginPageFunctionality.LoginPage_NavigateToLoginPageFromContact	details »
Test.dll: Test.Smoke.ITPortal_Login	details »
Test.dll: Test.Smoke.ITPortal_Logout	details »

Stacktraces:

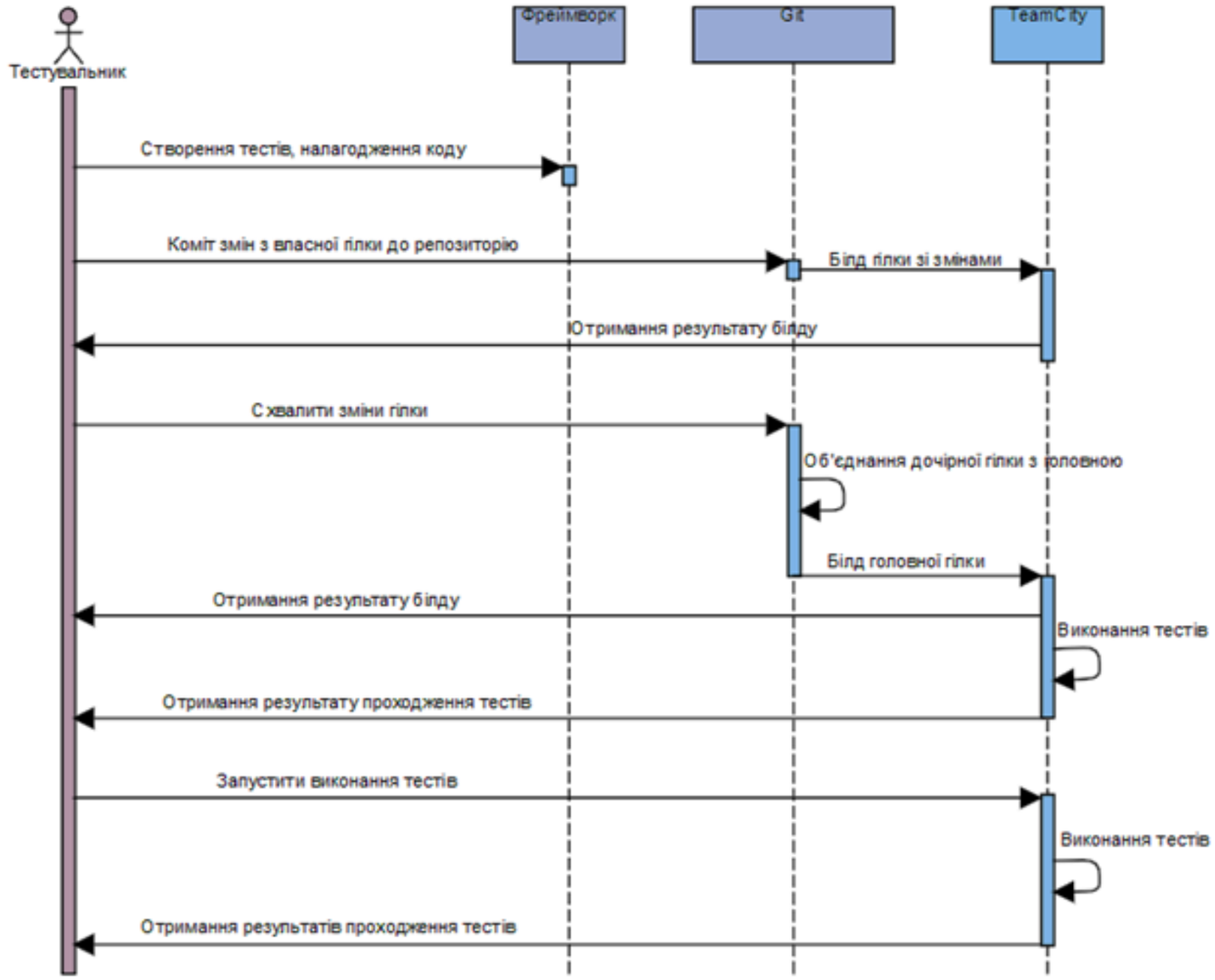
```
Test.dll: Test.LoginPageFunctionality.LoginPage_CollapseMenu
  Login Menu is not collapsed!
  Expected: True
  But was:  False

   at Test.LoginPageFunctionality.LoginPage_CollapseMenu() in
C:\BuildAgent\work\c13e7d8dcdccdce5\Test\LoginPageFunctionality.cs:line 52

------- Stdout: -------
[Info: 09:11:58 02.06.19] Click on Element for Button 'Expand Collapse Button'
(LoginHeaderSection.ExpandCollapseButton; JDI_Web.Selenium.Elements.APIInteract.GetElementModule)
Click on Element for Button 'Expand Collapse Button' (LoginHeaderSection.ExpandCollapseButton;
JDI_Web.Selenium.Elements.APIInteract.GetElementModule)
[Info: 09:11:59 02.06.19] Click on Element done
```

					ДП ІС-5115.1260-с.КЕ		
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	Літера	Маса / Масштаб
Розробив		Устенко Ю.Т.					
Перевірив		Новікова П.А.				Аркуш 1	Аркушів 1
Т. кон.							
Н. кон.		Тєлишева Т.О.			Система автоматизації тестування фреймворків Web-порталу	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-з51	
Затвердив		Новікова П.А.					

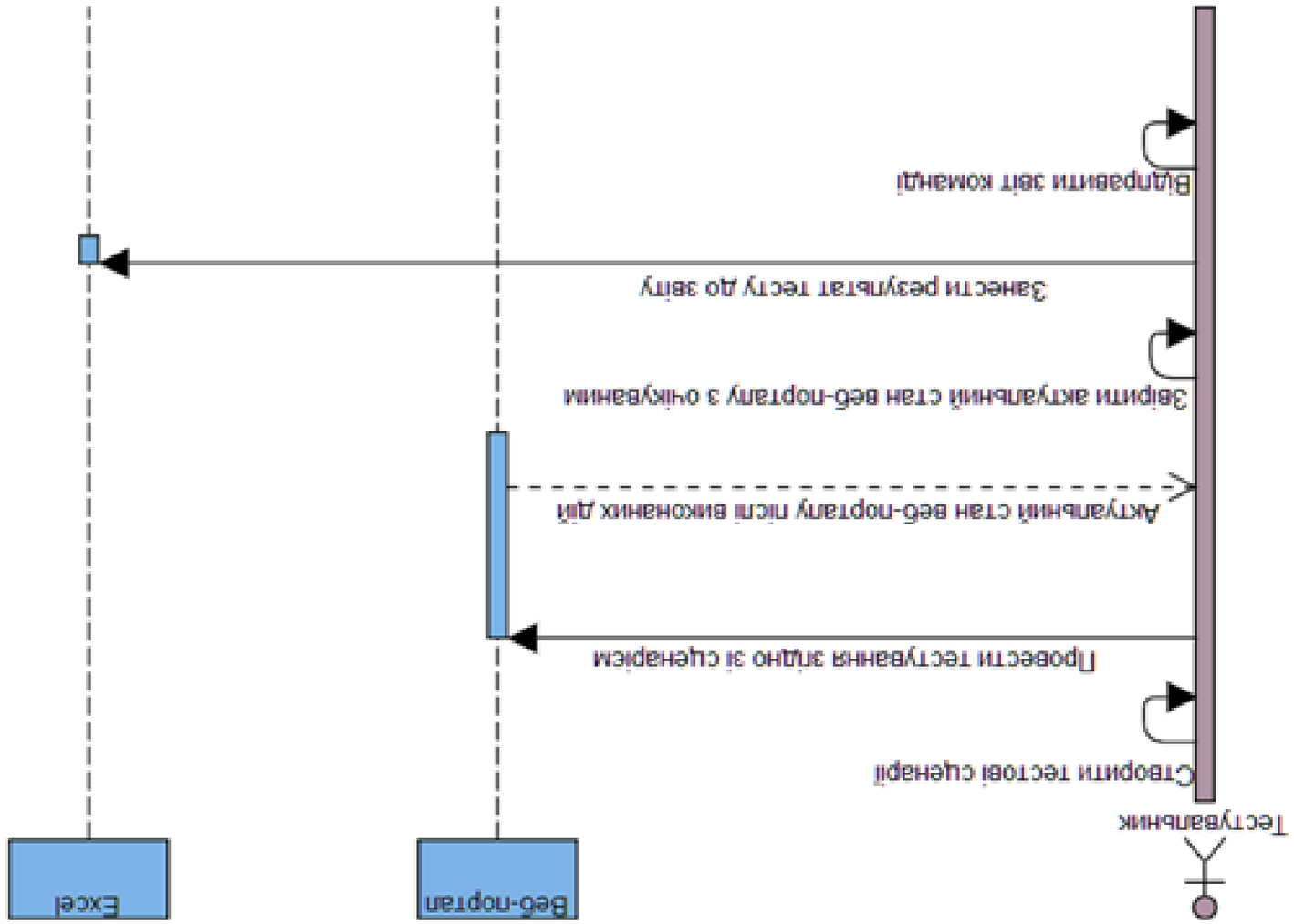
```xml
<?xml version="1.0" encoding="utf-8"?>
<ListExtendedUsersWithMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <User>
    <Role>Student</Role>
    <FirstName>First</FirstName>
    <LastName>Last Name</LastName>
    <Password>password1</Password>
    <Email />
    <Institution>institu</Institution>
    <Object>
      <Item1>Email</Item1>
      <Item2>Це поле є обов'язкове.</Item2>
    </Object>
  </User>
  <User>
    <Role>Student</Role>
    <FirstName />
    <LastName>Last Name</LastName>
    <Password>password1</Password>
    <Email>1test@test.com</Email>
    <Institution>institu</Institution>
    <Object>
      <Item1>FirstName</Item1>
      <Item2>Це поле є обов'язкове.</Item2>
    </Object>
  </User>
  <User>
    <Role>Student</Role>
    <FirstName>Third</FirstName>
    <LastName>Last Name</LastName>
    <Password>password2</Password>
    <Email>2test@test.com</Email>
    <Institution />
    <Object>
      <Item1>Institution</Item1>
      <Item2>Це поле є обов'язкове.</Item2>
    </Object>
  </User>
  <User>...</User>
  <User>...</User>
</ListExtendedUsersWithMessage>
```

```
public class ExtendedUser
{
    [Name("User role")]
    0 references | 0 changes | 0 authors, 0 changes
    public UserRole? Role { get; set; }

    [Name("First name")]
    1 reference | 0 changes | 0 authors, 0 changes
    public string FirstName { get; set; }

    [Name("Last name")]
    0 references | 0 changes | 0 authors, 0 changes
    public string LastName { get; set; }

    [Name("Password")]
    2 references | 0 changes | 0 authors, 0 changes
    public string Password { get; set; }

    [Name("Email")]
    1 reference | 0 changes | 0 authors, 0 changes
    public string Email { get; set; }

    [Name("Position")]
    0 references | 0 changes | 0 authors, 0 changes
    public string Position { get; set; }

    [Name("Institution")]
    0 references | 0 changes | 0 authors, 0 changes
    public string Institution { get; set; }

    4 references | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public (string,string) Object { get; set; }
}

2 references | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
public class ListExtendedUsersWithMessage : ITestDataSource
{
    [XmlElement(ElementName = "User")]
    2 references | Юлия Устенко, 4 days ago | 1 author, 1 change | 1 work item
    public ExtendedUser[] Users { get; set; }
}
```

					ДП ІС-5115.1260-с.КЕ	
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	Літера / Маса / Масштаб
Розробив		Устенко Ю.Т.				
Перевірив		Новікова П.А.				Аркуш 1 / Аркушів 1
Т. кон.						
Н. кон.		Тєлишева Т.О.			Система автоматизації тестування фреймворків Web-порталу	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-351
Затвердив		Новікова П.А.				



					ДП ІС-5115.1260-с.ССП		
						Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Схема послідовності роботи тестувальника у межах системи		
Розробив		Устенко Ю.Т.				Аркуш 1	Аркушів 1
Перевірив		Новікова П.А.					
Т. кон.							
Н. кон.		Тєлишева Т.О.			Система автоматизації тестування фреймворків Web-порталу	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-з51	
Затвердив		Новікова П.А.					



Зм.	Арк.	№ документа	Підпис	Дата	Схема послідовності роботи спеціаліста з ручного тестування	Літера	Маса	Масштаб
Розробив		Устенко Ю.Т.						
Перевірив		Новікова Л.А.						
Т. контр.						Аркуш 1		Аркушів 1
Н. контр.		Тепцова Т.О.			Система автоматизації тестування фрейлансерів Web-порталу	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-351		
Затвердив		Новікова Л.А.						

ДП ІС-5115.1260-с.ССП